2ª Edición

Inteligencia Artificial

Un Enfoque Moderno

PEARSON
Prentice Hall

Stuart Russell
Peter Norvig

INTELIGENCIA ARTIFICIAL

UN ENFOQUE MODERNO

Segunda edición

INTELIGENCIA ARTIFICIAL

UN ENFOQUE MODERNO

Segunda edición

Stuart J. Russell y Peter Norvig

Traducción:

Juan Manuel Corchado Rodríguez
Facultad de Ciencias
Universidad de Salamanca

Fernando Martín Rubio, José Manuel Cadenas Figueredo, Luis Daniel Hernández Molinero y Enrique Paniagua Arís
Facultad de Informática
Universidad de Murcia

Raquel Fuentetaja Pinzán y Mónica Robledo de los Santos
Universidad Pontificia de Salamanca, campus Madrid

Ramón Rizo Aldeguer
Escuela Politécnica Superior
Universidad de Alicante

Revisión técnica:

Juan Manuel Corchado Rodríguez
Facultad de Ciencias
Universidad de Salamanca

Fernando Martín Rubio
Facultad de Informática
Universidad de Murcia

Andrés Castillo Sanz y María Luisa Díez Plata
Facultad de Informática
Universidad Pontificia de Salamanca, campus Madrid

Coordinación general de la traducción y revisión técnica:

Luis Joyanes Aguilar
Facultad de Informática
Universidad Pontificia de Salamanca, campus Madrid

PEARSON
Prentice Hall

Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima
Montevideo • San Juan • San José • Santiago • São Paulo • White Plains

Datos de catalogación bibliográfica

RUSSELL, S. J.; NORVIG, P.

INTELIGENCIA ARTIFICIAL. UN ENFOQUE MODERNO
Segunda edición

PEARSON EDUCACIÓN, S.A., Madrid, 2004

ISBN: 978-84-205-4003-0
Materia: Informática 681.3

Formato 195 × 250 Páginas: 1240

Todos los derechos reservados.
Queda prohibida, salvo excepción prevista en la ley, cualquier forma de reproducción, distribución, comunicación pública y transformación de esta obra sin contar con autorización de los titulares de propiedad intelectual.
La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual *(arts. 270 y sgts. Código Penal).*

DERECHOS RESERVADOS
© 2004 por PEARSON EDUCACIÓN, S.A.
Ribera del Loira, 28
28042 Madrid (España)

INTELIGENCIA ARTIFICIAL. UN ENFOQUE MODERNO. Segunda edición
RUSSELL, S. J.; NORVIG, P.

ISBN: 978-84-205-4003-0
Depósito Legal: M-14511-2008
Última reimpresión: 2008

PEARSON PRENTICE HALL es un sello editorial autorizado de PEARSON EDUCACIÓN, S.A.

Authorized translation from the English language edition, entitled *ARTIFICIAL INTELLIGENCE: A MODERN APPROACH,* 2nd edition by RUSSELL, STUART; NORVIG, PETER.
Published by Pearson Education, Inc, publishing as Prentice Hall.
© 2003. All rights reserved.
No part or this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.
ISBN: 0-13-790395-2

Equipo editorial:
Editor: David Fayerman Aragón
Técnico editorial: Ana Isabel García Borro

Equipo de producción:
Director: José Antonio Clares
Técnico: José Antonio Hernán

Diseño de cubierta: Equipo de diseño de PEARSON EDUCACIÓN, S.A.

Composición: COPIBOOK, S.L.

Impreso por:

IMPRESO EN MÉXICO - PRINTED IN MEXICO

Contenido

Prólogo **XIX**

Sobre los autores **XXV**

1 Introducción **1**
1.1 ¿Qué es la IA? 2
Comportamiento humano: el enfoque de la Prueba de Turing 3
Pensar como un humano: el enfoque del modelo cognitivo 3
Pensamiento racional: el enfoque de las «leyes del pensamiento» 4
Actuar de forma racional: el enfoque del agente racional 5
1.2 Los fundamentos de la inteligencia artificial 6
Filosofía (desde el año 428 a.C. hasta el presente) 6
Matemáticas (aproximadamente desde el año 800 al presente) 9
Economía (desde el año 1776 hasta el presente) 11
Neurociencia (desde el año 1861 hasta el presente) 12
Psicología (desde el año 1879 hasta el presente) 14
Ingeniería computacional (desde el año 1940 hasta el presente) 16
Teoría de control y cibernética (desde el año 1948 hasta el presente) 17
Lingüística (desde el año 1957 hasta el presente) 18
1.3 Historia de la inteligencia artificial 19
Génesis de la inteligencia artificial (1943-1955) 19
Nacimiento de la inteligencia artificial (1956) 20
Entusiasmo inicial, grandes esperanzas (1952-1969) 21
Una dosis de realidad (1966-1973) 24
Sistemas basados en el conocimiento: ¿clave del poder? (1969-1979) 26
La IA se convierte en una industria (desde 1980 hasta el presente) 28
Regreso de las redes neuronales (desde 1986 hasta el presente) 29
IA se convierte en una ciencia (desde 1987 hasta el presente) 29
Emergencia de los sistemas inteligentes (desde 1995 hasta el presente) 31
1.4 El estado del arte 32
1.5 Resumen 33
Notas bibliográficas e históricas 34
Ejercicios 35

2 Agentes inteligentes **37**
2.1 Agentes y su entorno 37
2.2 Buen comportamiento: el concepto de racionalidad 40
Medidas de rendimiento 40
Racionalidad 41

Omnisciencia, aprendizaje y autonomía 42
2.3 La naturaleza del entorno 44
Especificación del entorno de trabajo 44
Propiedades de los entornos de trabajo 47
2.4 Estructura de los agentes 51
Programas de los agentes 51
Agentes reactivos simples 53
Agentes reactivos basados en modelos 55
Agentes basados en objetivos 57
Agentes basados en utilidad 58
Agentes que aprenden 59
2.5 Resumen 62
Notas bibliográficas e históricas 63
Ejercicios 65

3 Resolver problemas mediante búsqueda 67
3.1 Agentes resolventes-problemas 67
Problemas y soluciones bien definidos 70
Formular los problemas 71
3.2 Ejemplos de problemas 72
Problemas de juguete 73
Problemas del mundo real 76
3.3 Búsqueda de soluciones 78
Medir el rendimiento de la resolución del problema 80
3.4 Estrategias de búsqueda no informada 82
Búsqueda primero en anchura 82
Búsqueda de costo uniforme 84
Búsqueda primero en profundidad 85
Búsqueda de profundidad limitada 87
Búsqueda primero en profundidad con profundidad iterativa 87
Búsqueda bidireccional 89
Comparación de las estrategias de búsqueda no informada 91
3.5 Evitar estados repetidos 91
3.6 Búsqueda con información parcial 94
Problemas sin sensores 95
Problemas de contingencia 96
3.7 Resumen 97
Notas bibliográficas e históricas 98
Ejercicios 100

4 Búsqueda informada y exploración 107
4.1 Estrategias de búsqueda informada (heurísticas) 107
Búsqueda voraz primero el mejor 108
Búsqueda A*: minimizar el costo estimado total de la solución 110
Búsqueda heurística con memoria acotada 115
Aprender a buscar mejor 118
4.2 Funciones heurísticas 119
El efecto de la precisión heurística en el rendimiento 120
Inventar funciones heurísticas admisibles 121
Aprendizaje de heurísticas desde la experiencia 124
4.3 Algoritmos de búsqueda local y problemas de optimización 125

Búsqueda de ascensión de colinas 126
Búsqueda de temple simulado 129
Búsqueda por haz local 131
Algoritmos genéticos 131
4.4 Búsqueda local en espacios continuos 136
4.5 Agentes de búsqueda *online* y ambientes desconocidos 138
Problemas de búsqueda en línea (*online*) 138
Agentes de búsqueda en línea (*online*) 141
Búsqueda local en línea (*online*) 142
Aprendizaje en la búsqueda en línea (*online*) 144
4.6 Resumen 145
Notas bibliográficas e históricas 146
Ejercicios 151

5 Problemas de satisfacción de restricciones 155
5.1 Problemas de satisfacción de restricciones 155
5.2 Búsqueda con vuelta atrás para PSR 159
Variable y ordenamiento de valor 162
Propagación de la información a través de las restricciones 163
Comprobación hacia delante 163
Propagación de restricciones 164
Manejo de restricciones especiales 166
Vuelta atrás inteligente: mirando hacia atrás 167
5.3 Búsqueda local para problemas de satisfacción de restricciones 169
5.4 La estructura de los problemas 171
5.5 Resumen 175
Notas bibliográficas e históricas 176
Ejercicios 178

6 Búsqueda entre adversarios 181
6.1 Juegos 181
6.2 Decisiones óptimas en juegos 183
Estrategias óptimas 183
El algoritmo minimax 185
Decisiones óptimas en juegos multi-jugador 186
6.3 Poda alfa-beta 188
6.4 Decisiones en tiempo real imperfectas 191
Funciones de evaluación 192
Corte de la búsqueda 194
6.5 Juegos que incluyen un elemento de posibilidad 196
Evaluación de la posición en juegos con nodos de posibilidad 198
Complejidad del minimaxesperado 199
Juegos de cartas 200
6.6 Programas de juegos 202
6.7 Discusión 205
6.8 Resumen 207
Notas bibliográficas e históricas 208
Ejercicios 212

7 Agentes lógicos 217
7.1 Agentes basados en conocimiento 219
7.2 El mundo de *wumpus* 221

7.3 Lógica 224
7.4 Lógica proposicional: una lógica muy sencilla 229
Sintaxis 229
Semántica 230
Una base de conocimiento sencilla 233
Inferencia 233
Equivalencia, validez y *satisfacibilidad* 235
7.5 Patrones de razonamiento en lógica proposicional 236
Resolución 239
Forma normal conjuntiva 241
Un algoritmo de resolución 242
Completitud de la resolución 243
Encadenamiento hacia delante y hacia atrás 244
7.6 Inferencia proposicional efectiva 248
Un algoritmo completo con *backtracking* («vuelta atrás») 248
Algoritmos de búsqueda local 249
Problemas duros de *satisfacibilidad* 251
7.7 Agentes basados en lógica proposicional 253
Encontrar hoyos y *wumpus* utilizando la inferencia lógica 253
Guardar la pista acerca de la localización y la orientación del agente 255
Agentes basados en circuitos 256
Una comparación 260
7.8 Resumen 261
Notas bibliográficas e históricas 262
Ejercicios 266

8 Lógica de primer orden 271
8.1 Revisión de la representación 271
8.2 Sintaxis y semántica de la lógica de primer orden 277
Modelos en lógica de primer orden 277
Símbolos e interpretaciones 278
Términos 280
Sentencias atómicas 281
Sentencias compuestas 281
Cuantificadores 281
Cuantificador universal (∀) 282
Cuantificación existencial (∃) 283
Cuantificadores anidados 284
Conexiones entre ∀ y ∃ 285
Igualdad 286
8.3 Utilizar la lógica de primer orden 287
Aserciones y peticiones en lógica de primer orden 287
El dominio del parentesco 288
Números, conjuntos y listas 290
El mundo de *wumpus* 292
8.4 Ingeniería del conocimiento con lógica de primer orden 295
El proceso de ingeniería del conocimiento 296
El dominio de los circuitos electrónicos 297
Identificar la tarea 298
Recopilar el conocimiento relevante 298
Decidir el vocabulario 299

Codificar el conocimiento general del dominio 300
Codificar la instancia del problema específico 300
Plantear peticiones al procedimiento de inferencia 301
Depurar la base de conocimiento 301
8.5 Resumen 302
Notas bibliográficas e históricas 303
Ejercicios 304

9 Inferencia en lógica de primer orden 309
9.1 Lógica proposicional vs. Lógica de primer orden 310
Reglas de inferencia para cuantificadores 310
Reducción a la inferencia proposicional 311
9.2 Unificación y sustitución 312
Una regla de inferencia de primer orden 313
Unificación 314
Almacenamiento y recuperación 315
9.3 Encadenamiento hacia delante 318
Cláusulas positivas de primer orden 318
Un algoritmo sencillo de encadenamiento hacia delante 320
Encadenamiento hacia delante eficiente 322
Emparejar reglas con los hechos conocidos 322
Encadenamiento hacia delante incremental 324
Hechos irrelevantes 326
9.4 Encadenamiento hacia atrás 326
Un algoritmo de encadenamiento hacia atrás 327
Programación lógica 328
Implementación eficiente de programas lógicos 330
Inferencia redundante y bucles infinitos 332
Programación lógica con restricciones 334
9.5 Resolución 335
Formas normales conjuntivas en lógica de primer orden 336
La regla de inferencia de resolución 338
Demostraciones de ejemplo 338
Completitud de la resolución 341
Manejar la igualdad 345
Estrategias de resolución 346
Resolución unitaria 346
Resolución mediante conjunto soporte 347
Resolución lineal 347
Subsunción 347
Demostradores de teoremas 348
Diseño de un demostrador de teoremas 348
Ampliar el Prolog 349
Demostradores de teoremas como asistentes 350
Usos prácticos de los demostradores de teoremas 351
9.6 Resumen 352
Notas bibliográficas e históricas 353
Ejercicios 359

10 Representación del conocimiento 363
10.1 Ingeniería ontológica 363
10.2 Categoría y objetos 366

Objetos compuestos 368
Medidas 369
Sustancias y objetos 371
10.3 Acciones, situaciones y eventos 373
La ontología del cálculo de situaciones 373
Descripción de acciones en el cálculo de situaciones 375
Resolver el problema de la representación del marco 377
Resolver el problema de la inferencia del marco 379
El tiempo y el cálculo de eventos 380
Eventos generalizados 381
Procesos 383
Intervalos 384
Flujos y objetos 386
10.4 Eventos mentales y objetos mentales 387
Una teoría formal de creencias 387
Conocimiento y creencia 389
Conocimiento, tiempo y acción 390
10.5 El mundo de la compra por Internet 391
Comparación de ofertas 395
10.6 Sistemas de razonamiento para categorías 397
Redes semánticas 397
Lógica descriptiva 401
10.7 Razonamiento con información por defecto 402
Mundos abiertos y cerrados 403
Negación como fallo y semánticas de modelado estables 405
Circunscripción y lógica por defecto 406
10.8 Sistemas de mantenimiento de verdad 409
10.9 Resumen 411
Notas bibliográficas e históricas 412
Ejercicios 419

11 Planificación 427
11.1 El problema de planificación 428
El lenguaje de los problemas de planificación 429
Expresividad y extensiones 431
Ejemplo: transporte de carga aéreo 433
Ejemplo: el problema de la rueda de recambio 434
Ejemplo: el mundo de los bloques 434
11.2 Planificación con búsquedas en espacios de estado 436
Búsquedas hacia-delante en el espacio de estados 436
Búsquedas hacia-atrás en el espacio de estados 438
Heurísticas para la búsqueda en el espacio de estados 439
11.3 Planificación ordenada parcialmente 441
Ejemplo de planificación de orden parcial 445
Planificación de orden parcial con variables independientes 448
Heurísticas para planificación de orden parcial 449
11.4 Grafos de planificación 450
Grafos de planificación para estimación de heurísticas 453
El algoritmo GRAPHPLAN 454
Interrupción de GRAPHPLAN 457
11.5 Planificación con lógica proposicional 458

Descripción de problemas de planificación en lógica proposicional 458
Complejidad de codificaciones proposicionales 462
11.6 Análisis de los enfoques de planificación 463
11.7 Resumen 465
Notas bibliográficas e históricas 466
Ejercicios 469

12 Planificación y acción en el mundo real 475
12.1 Tiempo, planificación y recursos 475
Programación con restricción de recursos 478
12.2 Redes de planificación jerárquica de tareas 481
Representación de descomposición de acciones 482
Modificación de planificadores para su descomposición 484
Discusión 487
12.3 Planificación y acción en dominios no deterministas 490
12.4 Planificación condicional 493
Planificación condicional en entornos completamente observables 493
Planificación condicional en entornos parcialmente observables 498
12.5 Vigilancia de ejecución y replanificación 502
12.6 Planificación continua 507
12.7 Planificación multiagente 512
Cooperación: planes y objetivos conjuntos 512
Planificación condicional en entornos parcialmente observables 514
Mecanismos de coordinación 515
Mecanismos de coordinación 517
12.8 Resumen 517
Notas bibliográficas e históricas 518
Ejercicios 522

13 Incertidumbre 527
13.1 Comportamiento bajo incertidumbre 527
Manipulación del conocimiento incierto 528
Incertidumbre y decisiones racionales 530
Diseño de un agente de decisión teórico 531
13.2 Notación básica con probabilidades 532
Proposiciones 532
Sucesos atómicos 534
Probabilidad priori 534
Probabilidad condicional 536
13.3 Los axiomas de la probabilidad 537
Utilización de los axiomas de probabilidad 539
Por qué los axiomas de la probabilidad son razonables 540
13.4 Inferencia usando las distribuciones conjuntas totales 541
13.5 Independencia 544
13.6 La Regla de Bayes y su uso 546
Aplicación de la regla de Bayes: el caso sencillo 547
Utilización de la regla de Bayes: combinación de evidencia 548
13.7 El mundo *wumpus* revisado 550
13.8 Resumen 554
Notas bibliográficas e históricas 555
Ejercicios 557

14 Razonamiento probabilista **561**
14.1 La representación del conocimiento en un dominio incierto 561
14.2 La semántica de las redes bayesianas 564
La representación de la distribución conjunta completa 564
Un método para la construcción de redes bayesianas 565
Compactación y ordenación de nodos 566
Relaciones de independencia condicional en redes bayesianas 568
14.3 Representación eficiente de las distribuciones condicionales 569
Redes bayesianas con variables continuas 571
14.4 Inferencia exacta en redes bayesianas 574
Inferencia por enumeración 575
El algoritmo de eliminación de variables 577
La complejidad de la inferencia exacta 580
Algoritmos basados en grupos 580
14.5 Inferencia aproximada en redes bayesianas 581
Métodos de muestreo directo 582
Muestreo por rechazo en redes bayesianas 583
Ponderación de la verosimilitud 585
Inferencia por simulación en cadenas de Markov 587
14.6 Extensión de la probabilidad a representaciones de primer orden 590
14.7 Otros enfoques al razonamiento con incertidumbre 595
Métodos basados en reglas para razonamiento con incertidumbre 596
Representación de la ignorancia: teoría de Dempster-Shafer 598
Representación de la vaguedad: conjuntos difusos y lógica difusa 599
14.8. Resumen 601
Notas bibliográficas e históricas 601
Ejercicios 606

15 Razonamiento probabilista en el tiempo **611**
15.1 El tiempo y la incertidumbre 611
Estados y observaciones 612
Procesos estacionarios e hipótesis de Markov 613
15.2 Inferencia en modelos temporales 616
Filtrado y predicción 617
Suavizado 619
Encontrar la secuencia más probable 622
15.3 Modelos ocultos de Markov 624
Algoritmos matriciales simplificados 624
15.4 Filtros de Kalman 627
Actualización de distribuciones gaussianas 628
Un ejemplo unidimensional sencillo 629
El caso general 632
Aplicabilidad del filtrado de Kalman 633
15.5 Redes bayesianas dinámicas 635
Construcción de RBDs 636
Inferencia exacta en RBDs 640
Inferencia aproximada en RBDs 641
15.6 Reconocimiento del habla 645
Sonidos del habla 647
Palabras 649
Oraciones 651

Construcción de un reconocedor del habla 654
15.7 Resumen 656
Notas bibliográficas e históricas 656
Ejercicios 659

16 Toma de decisiones sencillas 663
16.1 Combinación de creencias y deseos bajo condiciones de incertidumbre 664
16.2 Los fundamentos de la teoría de la utilidad 665
Restricciones sobre preferencias racionales 666
... y entonces apareció la utilidad 668
16.3 Funciones de utilidad 669
La utilidad del dinero 669
Escalas de utilidad y evaluación de la utilidad 671
16.4 Funciones de utilidad multiatributo 674
Predominio 674
Estructura de preferencia y utilidad multiatributo 677
Preferencias sin incertidumbre 677
Preferencias con incertidumbre 678
16.5 Redes de decisión 679
Representación de un problema de decisión mediante una red de decisión 679
Evaluación en redes de decisión 681
16.6 El valor de la información 682
Un ejemplo sencillo 682
Una fórmula general 683
Propiedades del valor de la información 685
Implementación de un agente recopilador de información 685
16.7 Sistemas expertos basados en la teoría de la decisión 686
16.8 Resumen 690
Notas bibliográficas e históricas 690
Ejercicios 692

17 Toma de decisiones complejas 697
17.1 Problemas de decisión secuenciales 698
Un ejemplo 698
Optimalidad en problemas de decisión secuenciales 701
17.2 Iteración de valores 704
Utilidades de los estados 704
El algoritmo de iteración de valores 705
Convergencia de la iteración de valores 707
17.3 Iteración de políticas 710
17.4 Procesos de decisión de Markov parcialmente observables 712
17.5 Agentes basados en la teoría de la decisión 716
17.6 Decisiones con varios agentes: teoría de juegos 719
17.7 Diseño de mecanismos 729
17.8 Resumen 732
Notas bibliográficas e históricas 733
Ejercicios 736

18 Aprendizaje de observaciones 739
18.1 Formas de aprendizaje 739
18.2 Aprendizaje inductivo 742

18.3 Aprender árboles de decisión 744
Árboles de decisión como herramienta de desarrollo 744
Expresividad de los árboles de decisión 745
Inducir árboles de decisión a partir de ejemplos 746
Elección de los atributos de test 750
Valoración de la calidad del algoritmo de aprendizaje 752
Ruido y sobreajuste 753
Extensión de la aplicabilidad de los árboles de decisión 755
18.4 Aprendizaje de conjuntos de hipótesis 756
18.5 ¿Por qué funciona el aprendizaje?: teoría computacional del aprendizaje 760
¿Cuántos ejemplos se necesitan? 761
Aprendizaje de listas de decisión 763
Discusión 765
18.6 Resumen 766
Notas bibliográficas e históricas 767
Ejercicios 769

19 Conocimiento en el aprendizaje 773
19.1 Una formulación lógica del aprendizaje 773
Ejemplos e hipótesis 774
Búsqueda mejor-hipótesis-actual 776
Búsqueda de mínimo compromiso 778
19.2 Conocimiento en el aprendizaje 782
Algunos ejemplos sencillos 784
Algunos esquemas generales 784
19.3 Aprendizaje basado en explicaciones 786
Extraer reglas generales a partir de ejemplos 787
Mejorar la eficiencia 789
19.4 Aprendizaje basado en información relevante 791
Determinar el espacio de hipótesis 792
Aprender y utilizar información relevante 792
19.5 Programación lógica inductiva 795
Un ejemplo 795
Métodos de aprendizaje inductivo de arriba a abajo (*Top-down*) 798
Aprendizaje inductivo con deducción inversa 801
Hacer descubrimientos con la programación lógica inductiva 803
18.6 Resumen 805
Notas bibliográficas e históricas 806
Eercicios 809

20 Métodos estadísticos de aprendizaje 811
20.1 Aprendizaje estadístico 811
20.2 Aprendizaje con datos completos 815
Aprendizaje del parámetro de máxima verosimilitud: modelos discretos 815
Modelos de Bayes simples (Naive Bayes) 818
Aprendizaje de parámetros de máxima verosimilitud: modelos continuos 819
Aprendizaje de parámetros Bayesiano 821
Aprendizaje de la estructura de las redes bayesianas 823
20.3 Aprendizaje con variables ocultas: el algoritmo EM 825
Agrupamiento no supervisado: aprendizaje de mezclas de gaussianas 826
Aprendizaje de redes bayesianas con variables ocultas 829

Aprendizaje de modelos de Markov ocultos 831
Forma general del algoritmo EM 832
Aprendizaje de la estructura de las redes de Bayes con variables ocultas 833
20.4 Aprendizaje basado en instancias 834
Modelos de vecinos más cercanos 835
Modelos núcleo 837
20.5 Redes neuronales 838
Unidades en redes neuronales 839
Estructuras de las redes 840
Redes neuronales de una sola capa con alimentación-hacia-delante (perceptrones) . 842
Redes neuronales multicapa con alimentación hacia delante 846
Aprendizaje de la estructura de las redes neuronales 851
20.6 Máquinas núcleo 851
20.7 Caso de estudio: reconocedor de dígitos escritos a mano 855
20.8 Resumen 857
Notas bibliográficas e históricas 859
Ejercicios 863

21 Aprendizaje por refuerzo 867
21.1 Introducción 867
21.2 Aprendizaje por refuerzo pasivo 869
Estimación directa de la utilidad 870
Programación dinámica adaptativa 871
Aprendizaje de diferencia temporal 872
21.3 Aprendizaje por refuerzo activo 876
Exploración 876
Aprendizaje de una Función Acción-Valor 880
21.4 Generalización en aprendizaje por refuerzo 882
Aplicaciones a juegos 885
Aplicación a control de robots 886
21.5 Búsqueda de la política 887
21.6 Resumen 890
Notas bibliográficas e históricas 891
Ejercicios 894

22 La comunicación 897
22.1 La comunicación como acción 898
Fundamentos del lenguaje 899
Etapas de la comunicación 900
22.2 Una gramática formal para un fragmento del español 903
El léxico de e0 904
La Gramática de e0 904
22.3 Análisis sintáctico 905
Análisis sintáctico eficiente 908
22.4 Gramáticas aumentadas 914
Subcategorización del verbo 916
Capacidad generativa de las gramáticas aumentadas 918
22.5 Interpretación semántica 919
La semántica de un fragmento en español 920
Tiempo y forma verbal 921
Cuantificación 922

Interpretación pragmática 925
Generación de lenguajes con DCGs 926
22.6 Ambigüedad y desambigüedad 927
Desambiguación 929
22.7 Comprensión del discurso 930
Resolución por referencia 931
La estructura de un discurso coherente 932
22.8 Inducción gramatical 934
22.9 Resumen 936
Notas bibliográficas e históricas 937
Ejercicios 941

23 Procesamiento probabilístico del lenguaje 945
23.1 Modelos probabilísticos del lenguaje 945
Gramáticas probabilísticas independientes del contexto 949
Aprendizaje de probabilidades para PCFGs 950
Aprendizaje de la estructura de las reglas para PCFGs 951
23.2 Recuperación de datos 952
Evaluación de los Sistemas de RD 955
Refinamientos RD 956
Presentación de los conjuntos de resultados 957
Implementar sistemas RD 959
23.3 Extracción de la información 961
23.4 Traducción automática 964
Sistemas de traducción automáticos 966
Traducción automática estadística 967
Probabilidades de aprendizaje para la traducción automática 970
23.5 Resumen 972
Notas bibliográficas e históricas 972
Ejercicios 975

24 Percepción 979
24.1 Introducción 979
24.2 Formación de la imagen 981
Imágenes sin lentes: la cámara de orificio o pinhole 982
Sistemas de lentes 983
Luz: la fotometría de la formación de imágenes 983
Color: la espectrofotometría de la formación de imágenes 985
24.3 Operaciones de procesamiento de imagen a bajo nivel 986
Detección de aristas 987
Segmentación de la imagen 990
24.4 Extracción de información tridimensional 991
Movimiento 993
Estereoscopia binocular 996
Gradientes de textura 997
Sombreado 999
Contorno 1000
24.5 Reconocimiento de objetos 1004
Reconocimiento basado en la intensidad 1007
Reconocimiento basado en las características 1008
Estimación de postura 1010

24.6 Empleo de la visión para la manipulación y navegación 1012
24.7 Resumen 1014
Notas bibliográficas e históricas 1015
Ejercicios 1018

25 Robótica 1023
25.1 Introducción 1023
25.2 *Hardware* robótico 1025
Sensores 1025
Efectores 1027
25.3 Percepción robótica 1029
Localización 1031
Generación de mapas 1036
Otros tipos de percepción 1039
25.4 Planear el movimiento 1039
Espacio de configuración 1040
Métodos de descomposición en celdas 1043
Métodos de esqueletización 1046
25.5 Planificar movimientos inciertos 1047
Métodos robustos 1048
25.6 Movimiento 1051
Dinámica y control 1051
Control del campo de potencial 1054
Control reactivo 1055
25.7 Arquitecturas *software* robóticas 1057
Arquitectura de subsumpción 1058
Arquitectura de tres capas 1059
Lenguajes de programación robóticos 1060
25.8 Dominios de aplicación 1061
25.9 Resumen 1064
Notas bibliográficas e históricas 1065
Ejercicios 1069

26 Fundamentos filosóficos 1075
26.1 IA débil: ¿pueden las máquinas actuar con inteligencia? 1075
El argumento de incapacidad 1077
La objeción matemática 1078
El argumento de la informalidad 1079
26.2 IA fuerte: ¿pueden las máquinas pensar de verdad? 1081
El problema de mente-cuerpo 1084
El experimento del «cerebro en una cubeta» 1085
El experimento de la prótesis cerebral 1086
La habitación china 1088
26.3 La ética y los riesgos de desarrollar la Inteligencia Artificial 1090
26.4 Resumen 1095
Notas bibliográficas e históricas 1095
Ejercicios 1098

27 IA: presente y futuro 1099
27.1 Componentes de los agentes 1099
27.2 Arquitecturas de agentes 1102

27.3 ¿Estamos llevando la dirección adecuada? 1104
27.4 ¿Qué ocurriría si la IA tuviera éxito? 1106

A Fundamentos matemáticos 1109
A.1 Análisis de la complejidad y la notación O( ) 1109
Análisis asintótico 1109
Los problemas inherentemente difíciles y NP 1110
A.2 Vectores, matrices y álgebra lineal 1112
A.3 Distribuciones de probabilidades 1114
Notas bibliográficas e históricas 1115

B Notas sobre lenguajes y algoritmos 1117
B.1 Definición de lenguajes con Backus-Naur Form (BNF) 1117
B.2 Algoritmos de descripción en pseudocódigo 1118
B.3 Ayuda en línea 1119

Bibliografía 1121

Índice alfabético 1179

Prólogo

La Inteligencia Artificial (IA) es un campo grande (enorme), y este libro también. He intentado explorarlo con plena profundidad acompañándolo constantemente de lógica, probabilidad y matemáticas; de percepción, razonamiento, aprendizaje y acción, es decir, de todo lo que procede de los dispositivos microelectrónicos hasta los exploradores del planetario de la robótica. Otra razón para que este libro se pueda considerar espléndido es la profundidad en la presentación de los resultados, aunque nos hayamos esforzado por abarcar sólo las ideas más centrales en la parte principal de cada capítulo. En las notas bibliográficas al final de cada capítulo se proporcionan consejos para promover resultados.

El subtítulo de este libro es «Un Enfoque Moderno». La intención de esta frase bastante vacía es el hecho de que hemos intentado sintetizar lo que se conoce ahora dentro de un marco de trabajo común, en vez de intentar explicar cada uno de los subcampos de la IA dentro de su propio contexto histórico. Nos disculpamos ante aquellos cuyos subcampos son, como resultado, menos reconocibles de lo que podrían haber sido de cualquier otra forma.

El principal tema unificador es la idea del **agente inteligente**. Definimos la IA como el estudio de los agentes que reciben percepciones del entorno y llevan a cabo las acciones. Cada agente implementa una función la cual estructura las secuencias de las percepciones en acciones; también tratamos las diferentes formas de representar estas funciones, tales como sistemas de producción, agentes reactivos, planificadores condicionales en tiempo real, redes neurales y sistemas teóricos para las decisiones. Explicaremos el papel del aprendizaje cuando alcanza al diseñador y cómo se introduce en entornos desconocidos, mostrando también cómo ese papel limita el diseño del agente, favoreciendo así la representación y el razonamiento explícitos del conocimiento. Trataremos la robótica y su visión no como problemas con una definición independiente, sino como algo que ocurre para lograr los objetivos. Daremos importancia al entorno de las tareas al determinar el diseño apropiado de los agentes.

Nuestro objetivo principal es el de transmitir las *ideas* que han surgido durante los últimos 50 años de investigación en IA y trabajos afines durante los dos últimos milenios. Hemos intentado evitar una excesiva formalidad en la presentación de estas ideas a la vez que hemos intentado cuidar la precisión. Siempre que es necesario y adecuado, incluimos algoritmos en pseudo código para concretar las ideas, lo que se describe en el Apéndice B. Las implementaciones en varios lenguajes de programación están disponibles en el sitio Web de este libro, en la dirección de Internet **aima.cs.berkeley.edu**.

Este libro se ha pensado principalmente para utilizarse en un curso de diplomatura o en una serie de varios cursos. También se puede utilizar en un curso de licenciatura (quizás acompañado de algunas de las fuentes primarias, como las que se sugieren en las notas bibliográficas). Debida a la extensa cobertura y a la gran cantidad de algoritmos detallados, también es una herramienta útil como manual de referencia primario para alumnos de licenciatura superior y profesionales que deseen abarcar más allá de su propio subcampo. El único prerrequisito es familiarizarse con los conceptos básicos de la ciencia de la informática (algoritmos, estructuras de datos, complejidad) a un nivel de aprendizaje de segundo año. El cálculo de Freshman es útil para entender las redes neurales y el aprendizaje estadístico en detalle. En el Apéndice A se ofrecen los fundamentos matemáticos necesarios.

Visión general del libro

El libro se divide en ocho partes. La Parte I, **Inteligencia Artificial**, ofrece una visión de la empresa de IA basado en la idea de los agentes inteligentes, sistemas que pueden decidir qué hacer y entonces actutar. La Parte II, **Resolución de Problemas**, se concentra en los métodos para decidir qué hacer cuando se planean varios pasos con antelación, por ejemplo, navegar para cruzar un país o jugar al ajedrez. La Parte III, **Conocimiento y Razonamiento**, abarca las formas de representar el conocimiento sobre el mundo, es decir, cómo funciona, qué aspecto tiene actualmente y cómo podría actuar, y estudia también cómo razonar de forma lógica con ese conocimiento. La Parte IV, **Planificación**, abarca cómo utilizar esos métodos de razonamiento para decidir qué hacer, particularmente construyendo *planes*. La Parte V, **Conocimiento y Razonamiento Inciertos**, se asemeja a las Partes III y IV, pero se concentra en el razonamiento y en la toma de decisiones en presencia de la *incertidumbre* del mundo, la forma en que se podría enfrentar, por ejemplo, a un sistema de tratamiento y diagnosis médicos.

Las Partes II y V describen esa parte del agente inteligente responsable de alcanzar las decisiones. La Parte IV, **Aprendizaje**, describe los métodos para generar el conocimiento requerido por los componentes de la toma de decisiones. La Parte VII, **Comunicación, Percepción y Actuación**, describe las formas en que un agente inteligente puede percibir su entorno para saber lo que está ocurriendo, tanto si es mediante la visión, el tacto, el oído o el entendimiento del idioma; también describe cómo se pueden transformar los planes en acciones reales, o bien en movimientos de un robot, o bien en órdenes del lenguaje natural. Finalmente, la Parte VIII, **Conclusiones**, analiza el pasado y el futuro de la IA y sus repercusiones filosóficas y éticas.

Cambios de la primera edición

Desde que en 1995 se publicó la primera edición, la IA ha sufrido muchos cambios, por lo tanto esta edición también ha sufrido muchos cambios. Se han vuelto a escribir todos los capítulos elocuentemente para reflejar los últimos trabajos de este campo, reinterpretar los trabajos anteriores para que haya mayor coherencia con los actuales y mejorar la dirección pedagógica de las ideas. Los seguidores de la IA deberían estar alentados, ya que las técnicas actuales son mucho más prácticas que las del año 1995; por ejemplo, los algoritmos de planificación de la primera edición podrían generar planes sólo de unos cuantos pasos, mientras que los algoritmos de esta edición superan los miles de pasos. En la deducción probalística se han visto mejoras similares de órdenes de magnitud, procesamiento de lenguajes y otros subcampos. A continuación se muestran los cambios más destacables de esta edición:

- En la Parte I hacemos un reconocimiento de las contribuciones históricas a la teoría de controles, teoría de juegos, economía y neurociencia. Esto ayudará a sintonizar con una cobertura más integrada de estas ideas en los siguientes capítulos.
- En la Parte II se estudian los algoritmos de búsqueda en línea, y se ha añadido un capítulo nuevo sobre la satisfacción de las limitaciones. Este último proporciona una conexión natural con el material sobre la lógica.
- En la Parte III la lógica proposicional, que en la primera edición, se presentó como un peldaño a la lógica de primer orden, ahora se presenta como un lenguaje de representación útil por propio derecho, con rápidos algoritmos de deducción y diseños de agentes basados en circuitos. Los capítulos sobre la lógica de primer orden se han reorganizado para presentar el material de forma más clara, añadiendo el ejemplo del dominio de compras en Internet.
- En la Parte IV incluimos los métodos de planificación más actuales, tales como GRAPHPLAN y la planificación basada en la *satisfabilidad*, e incrementamos la cobertura de la planificación temporal, planificación condicional, planificación jerárquica y planificación multiagente.
- En la Parte V hemos aumentado el material de redes Bayesianas con algoritmos nuevos, tales como la eliminación de variables y el algoritmo Monte Carlo de cadena Markov; también hemos creado un capítulo nuevo sobre el razonamiento temporal incierto, abarcando los modelos ocultos de Markov, los filtros Kalman y las redes dinámicas Bayesianas. Los procesos de decisión de Markov se estudian en profundidad, añadiendo también secciones de teoría de juegos y diseños de mecanismos.
- En la Parte VI combinamos trabajos de aprendizaje estadístico, simbólico y neural, y añadimos secciones para impulsar los algoritmos, el algoritmo EM, aprendizaje basado en instancias, y métodos kernel «de núcleo» (soporte a máquinas vectoriales).
- En la Parte VII el estudio del procesamiento de lenguajes añade secciones sobre el procesamiento del discurso y la inducción gramatical, así como un capítulo so-

bre los modelos de lenguaje probalístico, con aplicaciones en la recuperación de información y en la traducción automática. El estudio de la robótica enfatiza la integración de datos inciertos de sensores, y el capítulo sobre la visión actualiza el material sobre el reconocimiento de objetos.
- En la Parte VIII introducimos una sección sobre las repercusiones éticas de la IA.

Utilización del libro

27 capítulos componen la totalidad del libro, que a su vez están compuestos por sesiones para clases que pueden durar una semana, por tanto para completar el libro se pueden necesitar hasta dos semestres. Alternativamente, se puede adaptar un curso de forma que se adecue a los intereses del instructor o del alumno. Si se utiliza el libro completo, servirá como soporte para cursos tanto si son reducidos, de introducción, para diplomaturas o para licenciados e ingenieros, de especialización, en temas avanzados. En la página Web aima.cs.berkeley.edu, se ofrecen programas de muestra recopilados de más de 600 escuelas y facultades, además de sugerencias que ayudarán a encontrar la secuencia más adecuada para sus necesidades.

Se han incluido 385 ejercicios que requieren una buena programación, y se han marcado con el icono de un teclado. Estos ejercicios se pueden resolver utilizando el repositorio de códigos que se encuentra en la página Web, aima.cs.berkeley.edu. Algunos de ellos son tan grandes que se pueden considerar proyectos de fin de trimestre. Algunos ejercicios requieren consultar otros libros de texto para investigar y se han marcado con el icono de un libro.

TÉRMINO NUEVO

Los puntos importantes también se han marcado con un icono que indica puntos importantes. Hemos incluido un índice extenso de 10.000 elementos que facilitan la utilización del libro. Además, siempre que aparece un **término nuevo**, también se destaca en el margen.

Utilización de la página Web

En la página Web se puede encontrar:

- implementaciones de los algoritmos del libro en diferentes lenguajes de programación,
- una relación de aproximadamente 600 universidades y centros, que han utilizado este libro, muchos de ellos con enlaces a los materiales de cursos en la red,
- una relación de unos 800 enlaces a sitios Web con contenido útil sobre la IA,
- una lista de todos los capítulos, uno a uno, material y enlaces complementarios,
- instrucciones sobre cómo unirse a un grupo de debate sobre el libro,
- instrucciones sobre cómo contactar con los autores mediante preguntas y comentarios,

- instrucciones sobre cómo informar de los errores del libro, en el caso probable de que existieran, y
- copias de las figuras del libro, junto con diapositivas y otro tipo de material para profesores.

Agradecimientos

Jitendra Malik ha escrito la mayor parte del Capítulo 24 (sobre la visión). Gran parte del Capítulo 25 (sobre robótica) de esta edición ha sido escrita por Sebastián Thrun, mientras que la primera edición fue escrita por John Canny. Doug Edwards investigó las notas históricas de la primera edición. Tim Huang, Mark Paskin y Cinthia Bruyns ayudó a realizar el formato de los diagramas y algoritmos. Alan Apt, Sondra Chavez, Toni Holm, Jake Warde, Irwin Zucker y Camille Trentacoste de Prentice Hall hicieron todo lo que pudieron para cumplir con el tiempo de desarrollo y aportaron muchas sugerencias sobre el diseño y el contenido del libro.

Stuart quiere agradecer a sus padres el continuo apoyo y aliento demostrado, y a su esposa, Loy Sheflott su gran paciencia e infinita sabiduría. También desea que Gordon y Lucy puedan leer esta obra muy pronto. Y a RUGS (Russell`s Unusual Group of Students) por haber sido de verdad de gran ayuda.

Peter quiere agradecer a sus padres (Torsten y Gerda) el ánimo que le aportaron para impulsarle a empezar el proyecto, y a su mujer Kris, hijos y amigos por animarle y tolerar todas las horas que ha dedicado en escribir y en volver a escribir su trabajo.

Estamos en deuda con los bibliotecarios de las universidades de Berkeley, Stanford, y con el MIT y la NASA; y con los desarrolladores de CiteSeer y Google por haber revolucionado la forma en que hemos realizado gran parte de la investigación.

Obviamente, nos es imposible agradecer a todas las personas que han utilizado el libro y que han realizado sugerencias, sin embargo nos gustaría mostrar especial agradecimiento a los comentarios que han realizado personas como Eyal Amir, Krzysztof Apt, Ellery Aziel, Jeff Van Baalen, Brian Baker, Don Barker, Tony Barrett, James Newton Bass, Don Beal, Howard Beck, Wolfgang Bibel, John Binder, Larry Bookman, David R. Boxall, Gerhard Brewka, Selmer Bringsjord, Carla Brodley, Chris Brown, Wilhelm Burger, Lauren Burka, Joao Cachopo, Murray Campbell, Norman Carver, Emmanuel Castro, Anil Chakravarthy, Dan Chisarick, Roberto Cipolla, David Cohen, James Coleman, Julie Ann Comparini, Gary Cottrell, Ernest Davis, Rina Dechter, Tom Dietterich, Chuck Dyer, Barbara Engelhardt, Doug Edwards, Kutluhan Erol, Oren Etzioni, Hana Filip, Douglas Fisher, Jeffrey Forbes, Ken Ford, John Fosler, Alex Franz, Bob Futrelle, Marek Galecki, Stefan Gerberding, Stuart Gill, Sabine Glesner, Seth Golub, Gosta Grahne, Russ Greiner, Eric Grimson, Barbara Grosz, Larry Hall, Steve Hanks, Othar Hansson, Ernst Heinz, Jim Hendler, Christoph Herrmann, Vasant Honavar, Tim Huang, Seth Hutchinson, Joost Jacob, Magnus Johansson, Dan Jurafsky, Leslie Kaelbling, Keiji Kanazawa, Surekha Kasibhatla, Simon Kasif, Henry Kautz, Gernot Kerschbaumer, Richard Kirby, Kevin Knight, Sven Koenig, Daphne Koller, Rich Korf, James Kurien, John Lafferty, Gus Larsson, John Lazzaro, Jon LeBlanc, Jason Leatherman, Frank Lee,

Edward Lim, Pierre Louveaux, Don Loveland, Sridhar Mahadevan, Jim Martin, Andy Mayer, David McGrane, Jay Mendelsohn, Brian Milch, Steve Minton, Vibhu Mittal, Leora Morgenstern, Stephen Muggleton, Kevin Murphy, Ron Musick, Sung Myaeng, Lee Naish, Pandu Nayak, Bernhard Nebel, Stuart Nelson, XuanLong Nguyen, Illah Nourbakhsh, Steve Omohundro, David Page, David Palmer, David Parkes, Ron Parr, Mark Paskin, Tony Passera, Michael Pazzani, Wim Pijls, Ira Pohl, Martha Pollack, David Poole, Bruce Porter, Malcolm Pradhan, Bill Pringle, Lorraine Prior, Greg Provan, William Rapaport, Philip Resnik, Francesca Rossi, Jonathan Schaeffer, Richard Scherl, Lars Schuster, Soheil Shams, Stuart Shapiro, Jude Shavlik, Satinder Singh, Daniel Sleator, David Smith, Bryan So, Robert Sproull, Lynn Stein, Larry Stephens, Andreas Stolcke, Paul Stradling, Devika Subramanian, Rich Sutton, Jonathan Tash, Austin Tate, Michael Thielscher, William Thompson, Sebastian Thrun, Eric Tiedemann, Mark Torrance, Randall Upham, Paul Utgoff, Peter van Beek, Hal Varian, Sunil Vemuri, Jim Waldo, Bonnie Webber, Dan Weld, Michael Wellman, Michael Dean White, Kamin Whitehouse, Brian Williams, David Wolfe, Bill Woods, Alden Wright, Richard Yen, Weixiong Zhang, Shlomo Zilberstein, y los revisores anónimos proporcionados por Prentice Hall.

Acerca de la portada

La ilustración de la portada ha sido diseñada por los autores y ejecutada por Lisa Marie Sardegna y Maryann Simmons utilizando SGI Inventor ™ y Photshop™ de Adobe, y representa los siguientes elementos de la historia de la IA:

1. El algoritmo de planificación del *De Motu Animalium*, Aristóteles (400 a.C.).
2. El generador de conceptos del *Ars Magna* de Ramón Lull (1300 d.C.).
3. El motor de diferencias de Charles Babbage, un prototipo del primer computador universal (1848).
4. La notación de lógica de primer orden de Gottlob Frege (1789).
5. Los diagramas del razonamiento lógico de Lewis Carroll (1886).
6. La notación de redes probalísticas de Sewall Wright (1921).
7. Alan Turing (1912-1954).
8. El Robot Shakey (1969-1973).
9. Un sistema experto de diagnóstico actual (1993).

Sobre los autores

Stuart Russell nació en 1962 en Portsmouth, Inglaterra. En 1982, obtuvo su B.A. en Física formando parte de los primeros en el cuadro de honor de la Universidad de Oxford. En 1986 obtuvo el Ph. D. en Informática por la Universidad de Stanford. Más tarde empezó a trabajar en la Universidad de California, Berkeley, en donde es profesor de Informática, director del centro de Sistemas Inteligentes y colaborador del Smith-Zadeh en Ingeniería. En 1990, recibió el premio Presidential Young Investigator Award de la National Sciencie Foundation (Fundación Nacional de las Ciencias), y en 1995 obtuvo el premio compartido en el Computers y Thought Award. En 1996 se convirtió en Millar Profesor de la University of California, y en el año 2000 fue nombrado rector de la universidad. En 1998, dio la lectura inaugural de la Stanford University, Forsythe Memorial Lectures. Forma parte como «Fellow», y es uno de los primeros miembros del Executive Council de la American Association for Artificical Intelligence. Ha publicado un gran número de trabajos, más de 100 sobre una gran variedad de temas en inteligencia artificial. Y es autor de otros dos libros: *The Use of Knowledge in Analogy and Induction*, y (con Erik Wefald) *Do the Right Thing: Studies in Limited Rationality*.

Peter Norvig es director de Search Quality de Google, Inc. y forma parte, como «Fellow» y miembro del Executive Council en la American Association for Artificial Intelligence. Anteriormente, fue director de la División de Ciencias Computacionales del Ames Research Center de la NASA, en donde supervisaba la investigación y el desarrollo en inteligencia artificial. Antes de eso, trabajó como director de ciencia en Junglee, donde ayudó a desarrollar uno de los primeros servicios de extracción de información en Internet, y trabajó también como científico senior en Sun Microsystems Laboratories, cuyo trabajo consistía en recuperar información inteligente. Recibió un B.S. en Matemáticas Aplicadas por la Brown University, y un Ph. D. en informática por la Universidad de California en Berkeley. Es profesor de la University of Southern California, y miembro de la facultad de investigación en Berkeley. Tiene en su haber más de 50 publicaciones en Informática entre las que se incluyen los libros: *Paradigms of AI Programming: Case Studies in Common Lisp, Verbmobil: A Translation System for Face-to-Face Dialog*, e *Intelligent Help Systems for UNIX*.

1 Introducción

Donde se intentará explicar por qué se considera a la inteligencia artificial un tema digno de estudio y donde se intentará definirla con exactitud; es esta tarea muy recomendable antes de emprender de lleno su estudio.

Los hombres se han denominado a sí mismos como *Homo sapiens* (hombre sabio) porque nuestras capacidades mentales son muy importantes para nosotros. Durante miles de años, hemos tratado de entender *cómo pensamos*; es decir, entender cómo un simple puñado de materia puede percibir, entender, predecir y manipular un mundo mucho más grande y complicado que ella misma. El campo de la **inteligencia artificial**, o IA, va más allá: no sólo intenta comprender, sino que también se esfuerza en construir entidades inteligentes.

INTELIGENCIA ARTIFICIAL

La IA es una de las ciencias más recientes. El trabajo comenzó poco después de la Segunda Guerra Mundial, y el nombre se acuñó en 1956. La IA se cita, junto a la biología molecular, como un campo en el que a la mayoría de científicos de otras disciplinas «les gustaría trabajar». Un estudiante de ciencias físicas puede pensar razonablemente que todas las buenas ideas han sido ya propuestas por Galileo, Newton, Einstein y otros. Por el contrario, la IA aún tiene flecos sin cerrar en los que podrían trabajar varios Einsteins a tiempo completo.

La IA abarca en la actualidad una gran variedad de subcampos, que van desde áreas de propósito general, como el aprendizaje y la percepción, a otras más específicas como el ajedrez, la demostración de teoremas matemáticos, la escritura de poesía y el diagnóstico de enfermedades. La IA sintetiza y automatiza tareas intelectuales y es, por lo tanto, potencialmente relevante para cualquier ámbito de la actividad intelectual humana. En este sentido, es un campo genuinamente universal.

1.1 ¿Qué es la IA?

Hemos proclamado que la IA es excitante, pero no hemos dicho qué *es*. La Figura 1.1 presenta definiciones de inteligencia artificial extraídas de ocho libros de texto. Las que aparecen en la parte superior se refieren a *procesos mentales* y al *razonamiento*, mientras que las de la parte inferior aluden a la *conducta*. Las definiciones de la izquierda miden el éxito en términos de la fidelidad en la forma de actuar de los *humanos*, mientras que las de la derecha toman como referencia un concepto ideal de inteligencia, que llamaremos **racionalidad**. Un sistema es racional si hace «lo correcto», en función de su conocimiento.

RACIONALIDAD

A lo largo de la historia se han seguido los cuatro enfoques mencionados. Como es de esperar, existe un enfrentamiento entre los enfoques centrados en los humanos y los centrados en torno a la racionalidad[1]. El enfoque centrado en el comportamiento humano debe ser una ciencia empírica, que incluya hipótesis y confirmaciones mediante experimentos. El enfoque racional implica una combinación de matemáticas e ingeniería. Cada grupo al mismo tiempo ha ignorado y ha ayudado al otro. A continuación revisaremos cada uno de los cuatro enfoques con más detalle.

Sistemas que piensan como humanos	Sistemas que piensan racionalmente
«El nuevo y excitante esfuerzo de hacer que los computadores piensen… máquinas con mentes, en el más amplio sentido literal». (Haugeland, 1985) «[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje…» (Bellman, 1978)	«El estudio de las facultades mentales mediante el uso de modelos computacionales». (Charniak y McDermott, 1985) «El estudio de los cálculos que hacen posible percibir, razonar y actuar». (Winston, 1992)
Sistemas que actúan como humanos	**Sistemas que actúan racionalmente**
«El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia». (Kurzweil, 1990) «El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor». (Rich y Knight, 1991)	«La Inteligencia Computacional es el estudio del diseño de agentes inteligentes». (Poole *et al*., 1998) «IA… está relacionada con conductas inteligentes en artefactos». (Nilsson, 1998)

Figura 1.1 Algunas definiciones de inteligencia artificial, organizadas en cuatro categorías.

[1] Conviene aclarar, que al distinguir entre comportamiento *humano* y *racional* no se está sugiriendo que los humanos son necesariamente «irracionales» en el sentido de «inestabilidad emocional» o «desequilibrio mental». Basta con darnos cuenta de que no somos perfectos: no todos somos maestros de ajedrez, incluso aquellos que conocemos todas las reglas del ajedrez; y desafortunadamente, no todos obtenemos un sobresaliente en un examen. Kahneman *et al*. (1982) ha elaborado un catálogo con algunos de los errores que sistemáticamente cometen los humanos cuando razonan.

Comportamiento humano: el enfoque de la Prueba de Turing

PRUEBA DE TURING

La **Prueba de Turing**, propuesta por Alan Turing (1950), se diseñó para proporcionar una definición operacional y satisfactoria de inteligencia. En vez de proporcionar una lista larga y quizá controvertida de cualidades necesarias para obtener inteligencia artificialmente, él sugirió una prueba basada en la incapacidad de diferenciar entre entidades inteligentes indiscutibles y seres humanos. El computador supera la prueba si un evaluador humano no es capaz de distinguir si las respuestas, a una serie de preguntas planteadas, son de una persona o no. En el Capítulo 26 se comentan detalles de esta prueba y se discute si un computador que supera la prueba es realmente inteligente. Hoy por hoy, podemos decir que programar un computador para que supere la prueba requiere un trabajo considerable. El computador debería poseer las siguientes capacidades:

PROCESAMIENTO DE LENGUAJE NATURAL

REPRESENTACIÓN DEL CONOCIMIENTO

RAZONAMIENTO AUTOMÁTICO

APRENDIZAJE MÁQUINA

- **Procesamiento de lenguaje natural** que le permita comunicarse satisfactoriamente en inglés.
- **Representación del conocimiento** para almacenar lo que se conoce o siente.
- **Razonamiento automático** para utilizar la información almacenada para responder a preguntas y extraer nuevas conclusiones.
- **Aprendizaje automático** para adaptarse a nuevas circunstancias y para detectar y extrapolar patrones.

PRUEBA DE TURING GLOBAL

La Prueba de Turing evitó deliberadamente la interacción *física* directa entre el evaluador y el computador, dado que para medir la inteligencia es innecesario simular físicamente a una persona. Sin embargo, la llamada Prueba Global de Turing incluye una señal de vídeo que permite al evaluador valorar la capacidad de percepción del evaluado, y también le da la oportunidad al evaluador de pasar objetos físicos «a través de una ventanita». Para superar la Prueba Global de Turing el computador debe estar dotado de

VISTA COMPUTACIONAL

ROBÓTICA

- **Visión computacional** para percibir objetos.
- **Robótica** para manipular y mover objetos.

Estas seis disciplinas abarcan la mayor parte de la IA, y Turing merece ser reconocido por diseñar una prueba que se conserva vigente después de 50 años. Los investigadores del campo de la IA han dedicado poco esfuerzo a la evaluación de sus sistemas con la Prueba de Turing, por creer que es más importante el estudio de los principios en los que se basa la inteligencia que duplicar un ejemplar. La búsqueda de un ingenio que «volara artificialmente» tuvo éxito cuando los hermanos Wright, entre otros, dejaron de imitar a los pájaros y comprendieron los principios de la aerodinámica. Los textos de ingeniería aerodinámica no definen el objetivo de su campo como la construcción de «máquinas que vuelen como palomas de forma que puedan incluso confundir a otras palomas».

Pensar como un humano: el enfoque del modelo cognitivo

Para poder decir que un programa dado piensa como un humano, es necesario contar con un mecanismo para determinar cómo piensan los humanos. Es necesario *penetrar* en el

funcionamiento de las mentes humanas. Hay dos formas de hacerlo: mediante introspección (intentando atrapar nuestros propios pensamientos conforme éstos van apareciendo) y mediante experimentos psicológicos. Una vez se cuente con una teoría lo suficientemente precisa sobre cómo trabaja la mente, se podrá expresar esa teoría en la forma de un programa de computador. Si los datos de entrada/salida del programa y los tiempos de reacción son similares a los de un humano, existe la evidencia de que algunos de los mecanismos del programa se pueden comparar con los que utilizan los seres humanos. Por ejemplo, a Allen Newell y Herbert Simon, que desarrollaron el «Sistema de Resolución General de Problemas» (SRGP) (Newell y Simon, 1961), no les bastó con que su programa resolviera correctamente los problemas propuestos. Lo que les interesaba era seguir la pista de las etapas del proceso de razonamiento y compararlas con las seguidas por humanos a los que se les enfrentó a los mismos problemas. En el campo interdisciplinario de la **ciencia cognitiva** convergen modelos computacionales de IA y técnicas experimentales de psicología intentando elaborar teorías precisas y verificables sobre el funcionamiento de la mente humana.

CIENCIA COGNITIVA

La ciencia cognitiva es un campo fascinante, merecedora de una enciclopedia dedicada a ella (Wilson y Keil, 1999). En este libro no se intenta describir qué se conoce de la cognición humana. Ocasionalmente se hacen comentarios acerca de similitudes o diferencias entre técnicas de IA y cognición humana. La auténtica ciencia cognitiva se fundamenta necesariamente en la investigación experimental en humanos y animales, y en esta obra se asume que el lector sólo tiene acceso a un computador para experimentar.

En los comienzos de la IA había confusión entre las distintas aproximaciones: un autor podría argumentar que un algoritmo resolvía adecuadamente una tarea y que *por tanto* era un buen modelo de representación humana, o viceversa. Los autores actuales hacen diferencia entre las dos reivindicaciones; esta distinción ha permitido que ambas disciplinas, IA y ciencia cognitiva, se desarrollen más rápidamente. Los dos campos continúan alimentándose entre sí, especialmente en las áreas de la visión y el lenguaje natural. En particular, el campo de la visión ha avanzado recientemente con la ayuda de una propuesta integrada que tiene en cuenta la evidencia neurofisiológica y los modelos computacionales.

Pensamiento racional: el enfoque de las «leyes del pensamiento»

El filósofo griego Aristóteles fue uno de los primeros en intentar codificar la «manera correcta de pensar», es decir, un proceso de razonamiento irrefutable. Sus **silogismos** son esquemas de estructuras de argumentación mediante las que siempre se llega a conclusiones correctas si se parte de premisas correctas (por ejemplo: «Sócrates es un hombre; todos los hombres son mortales; por lo tanto Sócrates es mortal»). Estas leyes de pensamiento supuestamente gobiernan la manera de operar de la mente; su estudio fue el inicio de un campo llamado **lógica**.

SILOGISMOS

LÓGICA

Estudiosos de la lógica desarrollaron, en el siglo XIX, una notación precisa para definir sentencias sobre todo tipo de elementos del mundo y especificar relaciones entre

ellos (compárese esto con la notación aritmética común, que prácticamente sólo sirve para representar afirmaciones acerca de la igualdad y desigualdad entre números). Ya en 1965 existían programas que, en principio, resolvían *cualquier* problema resoluble descrito en notación lógica[2]. La llamada tradición **logista** dentro del campo de la inteligencia artificial trata de construir sistemas inteligentes a partir de estos programas.

LOGISTA

Este enfoque presenta dos obstáculos. No es fácil transformar conocimiento informal y expresarlo en los términos formales que requieren de notación lógica, particularmente cuando el conocimiento que se tiene es inferior al 100 por 100. En segundo lugar, hay una gran diferencia entre poder resolver un problema «en principio» y hacerlo en la práctica. Incluso problemas con apenas una docena de datos pueden agotar los recursos computacionales de cualquier computador a menos que cuente con alguna directiva sobre los pasos de razonamiento que hay que llevar a cabo primero. Aunque los dos obstáculos anteriores están presentes en *todo* intento de construir sistemas de razonamiento computacional, surgieron por primera vez en la tradición lógica.

Actuar de forma racional: el enfoque del agente racional

AGENTE

Un **agente** es algo que razona (*agente* viene del latín *agere*, hacer). Pero de los agentes informáticos se espera que tengan otros atributos que los distingan de los «programas» convencionales, como que estén dotados de controles autónomos, que perciban su entorno, que persistan durante un período de tiempo prolongado, que se adapten a los cambios, y que sean capaces de alcanzar objetivos diferentes. Un **agente racional** es aquel que actúa con la intención de alcanzar el mejor resultado o, cuando hay incertidumbre, el mejor resultado esperado.

AGENTE RACIONAL

En el caso del enfoque de la IA según las «leyes del pensamiento», todo el énfasis se pone en hacer inferencias correctas. La obtención de estas inferencias correctas puede, a veces, formar *parte* de lo que se considera un agente racional, ya que una manera racional de actuar es llegar a la conclusión lógica de que si una acción dada permite alcanzar un objetivo, hay que llevar a cabo dicha acción. Sin embargo, el efectuar una inferencia correcta no depende siempre de la *racionalidad*, ya que existen situaciones para las que no hay nada correcto que hacer y en las que hay que tomar una decisión. Existen también formas de actuar racionalmente que no implican realizar inferencias. Por ejemplo, el retirar la mano de una estufa caliente es un acto reflejo mucho más eficiente que una respuesta lenta llevada a cabo tras una deliberación cuidadosa.

Todas la habilidades que se necesitan en la Prueba de Turing deben permitir emprender acciones racionales. Por lo tanto, es necesario contar con la capacidad para representar el conocimiento y razonar basándonos en él, porque ello permitirá alcanzar decisiones correctas en una amplia gama de situaciones. Es necesario ser capaz de generar sentencias comprensibles en lenguaje natural, ya que el enunciado de tales oraciones permite a los agentes desenvolverse en una sociedad compleja. El aprendizaje no se lleva a cabo por erudición exclusivamente, sino que profundizar en el conocimiento de cómo funciona el mundo facilita la concepción de estrategias mejores para manejarse en él.

[2] Si no se encuentra una solución, el programa nunca debe parar de buscarla.

La percepción visual es necesaria no sólo porque ver es divertido, sino porque es necesaria para poder tener una idea mejor de lo que una acción puede llegar a representar, por ejemplo, el ver un delicioso bocadillo contribuirá a que nos acerquemos a él.

Por esta razón, el estudiar la IA desde el enfoque del diseño de un agente racional ofrece al menos dos ventajas. La primera es más general que el enfoque que proporcionan las «leyes del pensamiento», dado que el efectuar inferencias correctas es sólo uno de los mecanismos existentes para garantizar la racionalidad. La segunda es más afín a la forma en la que se ha producido el avance científico que los enfoques basados en la conducta o pensamiento humano, porque la norma de la racionalidad está claramente definida y es de aplicación general. Por el contrario, la conducta humana se adapta bien a un entorno específico, y en parte, es producto de un proceso evolutivo complejo, en gran medida desconocido, que aún está lejos de llevarnos a la perfección. *Por tanto, esta obra se centrará en los principios generales que rigen a los agentes racionales y en los elementos necesarios para construirlos*. Más adelante quedará patente que a pesar de la aparente facilidad con la que se puede describir un problema, cuando se intenta resolver surgen una enorme variedad de cuestiones. El Capítulo 2 revisa algunos de estos aspectos con más detalle.

Un elemento importante a tener en cuenta es el siguiente: más bien pronto que tarde se observará cómo obtener una racionalidad perfecta (hacer siempre lo correcto) no es posible en entornos complejos. La demanda computacional que esto implica es demasiado grande. En la mayor parte de esta obra se adoptará la hipótesis de trabajo de que la racionalidad perfecta es un buen punto de partida para el análisis. Lo cual simplifica el problema y proporciona el escenario base adecuado sobre el que se asientan los cimientos de este campo. Los Capítulos 6 y 17 se centran explícitamente en el tema de la **racionalidad limitada** (actuar adecuadamente cuando no se cuenta con el tiempo suficiente para efectuar todos los cálculos que serían deseables).

RACIONALIDAD LIMITADA

1.2 Los fundamentos de la inteligencia artificial

Esta sección presenta una breve historia de las disciplinas que han contribuido con ideas, puntos de vista y técnicas al desarrollo de la IA. Como toda revisión histórica, en este caso se centra en un pequeño número de personas, eventos e ideas e ignora otras que también fueron importantes. La historia se organiza en torno a una serie de cuestiones, dejando claro que no se quiere dar la impresión de que estas cuestiones son las únicas por las que las disciplinas se han preocupado y que el objetivo último de todas estas disciplinas era hacer avanzar la IA.

Filosofía (desde el año 428 a.C. hasta el presente)

- ¿Se pueden utilizar reglas formales para extraer conclusiones válidas?
- ¿Cómo se genera la inteligencia mental a partir de un cerebro físico?
- ¿De dónde viene el conocimiento?
- ¿Cómo se pasa del conocimiento a la acción?

Aristóteles (384-322 a.C.) fue el primero en formular un conjunto preciso de leyes que gobernaban la parte racional de la inteligencia. Él desarrolló un sistema informal para razonar adecuadamente con silogismos, que en principio permitía extraer conclusiones mecánicamente, a partir de premisas iniciales. Mucho después, Ramón Lull (d. 1315) tuvo la idea de que el razonamiento útil se podría obtener por medios artificiales. Sus «ideas» aparecen representadas en la portada de este manuscrito. Thomas Hobbes (1588-1679) propuso que el razonamiento era como la computación numérica, de forma que «nosotros sumamos y restamos silenciosamente en nuestros pensamientos». La automatización de la computación en sí misma estaba en marcha; alrededor de 1500, Leonardo da Vinci (1452-1519) diseñó, aunque no construyó, una calculadora mecánica; construcciones recientes han mostrado que su diseño era funcional. La primera máquina calculadora conocida se construyó alrededor de 1623 por el científico alemán Wilhelm Schickard (1592-1635), aunque la Pascalina, construida en 1642 por Blaise Pascal (1623-1662), sea más famosa. Pascal escribió que «la máquina aritmética produce efectos que parecen más similares a los pensamientos que a las acciones animales». Gottfried Wilhelm Leibniz (1646-1716) construyó un dispositivo mecánico con el objetivo de llevar a cabo operaciones sobre conceptos en lugar de sobre números, pero su campo de acción era muy limitado.

Ahora que sabemos que un conjunto de reglas pueden describir la parte racional y formal de la mente, el siguiente paso es considerar la mente como un sistema físico. René Descartes (1596-1650) proporciona la primera discusión clara sobre la distinción entre la mente y la materia y los problemas que surgen. Uno de los problemas de una concepción puramente física de la mente es que parece dejar poco margen de maniobra al libre albedrío: si el pensamiento está totalmente gobernado por leyes físicas, entonces una piedra podría «decidir» caer en dirección al centro de la Tierra gracias a su libre albedrío. A pesar de ser denodado defensor de la capacidad de razonamiento, Descartes fue un defensor del **dualismo**. Sostenía que existe una parte de la mente (o del alma o del espíritu) que está al margen de la naturaleza, exenta de la influencia de las leyes físicas. Los animales, por el contrario, no poseen esta cualidad dual; a ellos se le podría concebir como si se tratasen de máquinas. Una alternativa al dualismo es el **materialismo**, que considera que las operaciones del cerebro realizadas de acuerdo a las leyes de la física *constituyen* la mente. El libre albedrío es simplemente la forma en la que la percepción de las opciones disponibles aparecen en el proceso de selección.

DUALISMO

MATERIALISMO

Dada una mente física que gestiona conocimiento, el siguiente problema es establecer las fuentes de este conocimiento. El movimiento **empírico**, iniciado con el *Novum Organum*[3], de Francis Bacon (1561-1626), se caracteriza por el aforismo de John Locke (1632-1704): «Nada existe en la mente que no haya pasado antes por los sentidos». David Hume (1711-1776) propuso en *A Treatise of Human Nature* (Hume, 1739) lo que actualmente se conoce como principio de **inducción**: las reglas generales se obtienen mediante la exposición a asociaciones repetidas entres sus elementos. Sobre la base de las propuestas de Ludwig Wittgenstein (1889-1951) y Bertrand Russell (1872-1970), el famoso Círculo de Viena, liderado por Rudolf Carnap (1891-1970), desarrolló la doctrina del **positivismo lógico**. Esa doctrina sostiene que todo el conocimiento se puede

EMPÍRICO

INDUCCIÓN

[3] Una actualización del *Organon*, o instrumento de pensamiento, de Aristóteles.

POSITIVISMO LÓGICO

SENTENCIA DE OBSERVACIÓN

TEORÍA DE LA CONFIRMACIÓN

caracterizar mediante teorías lógicas relacionadas, en última instancia, con **sentencias de observación** que corresponden a estímulos sensoriales[4]. La **teoría de la confirmación** de Carnap y Carl Hempel (1905-1997) intenta explicar cómo el conocimiento se obtiene a partir de la experiencia. El libro de Carnap, *The Logical Structure of the World* (1928), define un procedimiento computacional explícito para la extracción de conocimiento a partir de experiencias primarias. Fue posiblemente la primera teoría en mostrar la mente como un proceso computacional.

El último elemento en esta discusión filosófica sobre la mente es la relación que existe entre conocimiento y acción. Este asunto es vital para la IA, ya que la inteligencia requiere tanto acción como razonamiento. Más aún, simplemente con comprender cómo se justifican determinadas acciones se puede llegar a saber cómo construir un agente cuyas acciones sean justificables (o racionales). Aristóteles argumenta que las acciones se pueden justificar por la conexión lógica entre los objetivos y el conocimiento de los efectos de las acciones (la última parte de este extracto también aparece en la portada de este libro):

> ¿Cómo es que el pensamiento viene acompañado en algunos casos de acciones y en otros no?, ¿en algunos casos por movimiento y en otros no? Parece como si la misma cosa sucediera tanto si razonáramos o hiciéramos inferencias sobre objetos que no cambian; pero en este caso el fin es una proposición especulativa... mientras la conclusión resultante de las dos premisas es una acción... Yo necesito abrigarme; una manta abriga. Yo necesito una manta. Qué necesito, qué debo hacer; necesito una manta. Necesito hacer una manta. Y la conclusión, «Yo tengo que hacer una manta», es una acción. (Nussbaum, 1978, p. 40)

En *Nicomachean Ethics* (Libro III. 3, 1112b), Aristóteles continúa trabajando en este tema, sugiriendo un algoritmo:

> Nosotros no reflexionamos sobre los fines, sino sobre los medios. Un médico no reflexiona sobre si debe curar, ni un orador sobre si debe persuadir... Ellos asumen el fin y consideran cómo y con qué medios se obtienen, y si resulta fácil y es por tanto productivo; mientras que si sólo se puede alcanzar por un medio se tiene en consideración *cómo* se alcanzará por este y por qué medios se obtendrá *éste*, hasta que se llegue a la causa primera..., y lo último en el orden del análisis parece ser lo primero en el orden de los acontecimientos. Y si se llega a un estado imposible, se abandona la búsqueda, como por ejemplo si se necesita dinero y no se puede conseguir; pero si hay una posibilidad se intentará.

El algoritmo de Aristóteles se implementó 2.300 años más tarde por Newell y Simon con la ayuda de su programa SRGP. El cual se conoce como sistema de planificación regresivo (*véase* el Capítulo 11).

El análisis basado en objetivos es útil, pero no indica qué hacer cuando varias acciones nos llevan a la consecución del objetivo, o cuando ninguna acción facilita su completa consecución. Antoine Arnauld (1612-1694) describió correctamente una forma cuantitativa para decidir qué acción llevar a cabo en un caso como este (*véase* el Capítulo 16). El libro *Utilitarianism* (Mill, 1863) de John Stuart Mill (1806-1873) propone

[4] En este contexto, es posible comprobar o rechazar toda aseveración significativa mediante el análisis del significado de las palabras o mediante la experimentación. Dado que esto no es aplicable en la mayor parte del ámbito de la metafísica, como era intención, el positivismo lógico se hizo impopular en algunos círculos.

la idea de un criterio de decisión racional en todos los ámbitos de la actividad humana. En la siguiente sección se explica una teoría de la decisión más formalmente.

Matemáticas (aproximadamente desde el año 800 al presente)

- ¿Qué reglas formales son las adecuadas para obtener conclusiones válidas?
- ¿Qué se puede computar?
- ¿Cómo razonamos con información incierta?

Los filósofos delimitaron las ideas más importantes de la IA, pero para pasar de ahí a una ciencia formal es necesario contar con una formulación matemática en tres áreas fundamentales: lógica, computación y probabilidad.

El concepto de lógica formal se remonta a los filósofos de la antigua Grecia (*véase* el Capítulo 7), pero su desarrollo matemático comenzó realmente con el trabajo de George Boole (1815-1864) que definió la lógica proposicional o Booleana (Boole, 1847). En 1879, Gottlob Frege (1848-1925) extendió la lógica de Boole para incluir objetos y relaciones, y creó la lógica de primer orden que se utiliza hoy como el sistema más básico de representación de conocimiento[5]. Alfred Tarski (1902-1983) introdujo una teoría de referencia que enseña cómo relacionar objetos de una lógica con objetos del mundo real. El paso siguiente consistió en definir los límites de lo que se podía hacer con la lógica y la informática.

ALGORITMO

Se piensa que el primer **algoritmo** no trivial es el algoritmo Euclídeo para el cálculo del máximo común divisor. El considerar los algoritmos como objetos en sí mismos se remonta a la época de al-Khowarazmi, un matemático persa del siglo IX, con cuyos escritos también se introdujeron los números arábigos y el álgebra en Europa. Boole, entre otros, presentó algoritmos para llevar a cabo deducciones lógicas y hacia el final del siglo XIX se llevaron a cabo numerosos esfuerzos para formalizar el razonamiento matemático general con la lógica deductiva. En 1900, David Hilbert (1862-1943) presentó una lista de 23 problemas que acertadamente predijo ocuparían a los matemáticos durante todo ese siglo. En el último de ellos se preguntaba si existe un algoritmo que permita determinar la validez de cualquier proposición lógica en la que aparezcan números naturales (el famoso *Entscheidungsproblem*, o problema de decisión). Básicamente, lo que Hilbert se preguntaba es si hay límites fundamentales en la capacidad de los procedimientos efectivos de demostración. En 1930, Kurt Gödel (1906-1978) demostró que existe un procedimiento eficiente para demostrar cualquier aseveración verdadera en la lógica de primer orden de Frege y Russell, sin embargo con la lógica de primer orden no era posible capturar el principio de inducción matemática necesario para la caracterización de los números naturales. En 1931, demostró que, en efecto, existen límites reales. Mediante su **teorema de incompletitud** demostró que en cualquier lenguaje que tuviera la capacidad suficiente para expresar las propiedades de los números naturales, existen aseveraciones verdaderas no decidible en el sentido de que no es posible decidir su validez mediante ningún algoritmo.

TEOREMA DE INCOMPLETITUD

[5] La notación para la lógica de primer orden propuesta por Frege no se ha aceptado universalmente, por razones que son aparentemente obvias cuando se observa el ejemplo que aparece en la cubierta de este libro.

El resultado fundamental anterior se puede interpretar también como la indicación de que existen algunas funciones de los números enteros que no se pueden representar mediante un algoritmo, es decir no se pueden calcular. Lo anterior llevó a Alan Turing (1912-1954) a tratar de caracterizar exactamente aquellas funciones que sí *eran* susceptibles de ser caracterizadas. La noción anterior es de hecho problemática hasta cierto punto, porque no es posible dar una definición formal a la noción de cálculo o procedimiento efectivo. No obstante, la tesis de Church-Turing, que afirma que la máquina de Turing (Turing, 1936) es capaz de calcular cualquier función computable, goza de aceptación generalizada ya que proporciona una definición suficiente. Turing también demostró que existen algunas funciones que no se pueden calcular mediante la máquina de Turing. Por ejemplo, ninguna máquina puede decidir *en general* si un programa dado producirá una respuesta a partir de unas entradas, o si seguirá calculando indefinidamente.

INTRATABILIDAD

Si bien ser no decidible ni computable son importantes para comprender el proceso del cálculo, la noción de **intratabilidad** tuvo repercusiones más importantes. En términos generales se dice que un problema es intratable si el tiempo necesario para la resolución de casos particulares de dicho problema crece exponencialmente con el tamaño de dichos casos. La diferencia entre crecimiento polinomial y exponencial de la complejidad se destacó por primera vez a mediados de los años 60 (Cobham, 1964; Edmonds, 1965). Es importante porque un crecimiento exponencial implica la imposibilidad de resolver casos moderadamente grandes en un tiempo razonable. Por tanto, se debe optar por dividir el problema de la generación de una conducta inteligente en subproblemas que sean tratables en vez de manejar problemas intratables.

NP-COMPLETITUD

¿Cómo se puede reconocer un problema intratable? La teoría de la **NP-completitud**, propuesta por primera vez por Steven Cook (1971) y Richard Karp (1972) propone un método. Cook y Karp demostraron la existencia de grandes clases de problemas de razonamiento y búsqueda combinatoria canónica que son NP completos. Toda clase de problema a la que la clase de problemas NP completos se pueda reducir será seguramente intratable (aunque no se ha demostrado que los problemas NP completos son necesariamente intratables, la mayor parte de los teóricos así lo creen). Estos resultados contrastan con el optimismo con el que la prensa popular recibió a los primeros computadores, «Supercerebros Electrónicos» que eran «¡Más rápidos que Einstein!». A pesar del rápido incremento en la velocidad de los computadores, los sistemas inteligentes se caracterizarán por el uso cuidadoso que hacen de los recursos. De manera sucinta, ¡el mundo es un ejemplo de problema *extremadamente* grande! Recientemente la IA ha ayudado a explicar por qué algunos ejemplos de problemas NP completos son difíciles de resolver y otros son fáciles (Cheeseman *et al*., 1991).

PROBABILIDAD

Además de la lógica y el cálculo, la tercera gran contribución de las matemáticas a la IA es la teoría de la **probabilidad**. El italiano Gerolamo Cardano (1501-1576) fue el primero en proponer la idea de probabilidad, presentándola en términos de los resultados de juegos de apuesta. La probabilidad se convirtió pronto en parte imprescindible de las ciencias cuantitativas, ayudando en el tratamiento de mediciones con incertidumbre y de teorías incompletas. Pierre Fermat (1601-1665), Blaise Pascal (1623-1662), James Bernoulli (1654-1705), Pierre Laplace (1749-1827), entre otros, hicieron avanzar esta teoría e introdujeron nuevos métodos estadísticos. Thomas Bayes (1702-1761) propuso una

regla para la actualización de probabilidades subjetivas a la luz de nuevas evidencias. La regla de Bayes y el área resultante llamado análisis Bayesiano conforman la base de las propuestas más modernas que abordan el razonamiento incierto en sistemas de IA.

Economía (desde el año 1776 hasta el presente)

- ¿Cómo se debe llevar a cabo el proceso de toma de decisiones para maximizar el rendimiento?
- ¿Cómo se deben llevar a cabo acciones cuando otros no colaboren?
- ¿Cómo se deben llevar a cabo acciones cuando los resultados se obtienen en un futuro lejano?

La ciencia de la economía comenzó en 1776, cuando el filósofo escocés Adam Smith (1723-1790) publicó *An Inquiri into the Nature and Causes of the Wealth of Nations*. Aunque los antiguos griegos, entre otros, habían hecho contribuciones al pensamiento económico, Smith fue el primero en tratarlo como una ciencia, utilizando la idea de que las economías pueden concebirse como un conjunto de agentes individuales que intentan maximizar su propio estado de bienestar económico. La mayor parte de la gente cree que la economía sólo se trata de dinero, pero los economistas dicen que ellos realmente estudian cómo la gente toma decisiones que les llevan a obtener los beneficios esperados. Léon Walras (1834-1910) formalizó el tratamiento matemático del «beneficio deseado» o **utilidad,** y fue posteriormente mejorado por Frank Ramsey (1931) y después por John von Neumann y Oskar Morgenstern en su libro *The Theory of Games and Economic Behavior* (1944).

TEORÍA DE LA DECISIÓN

La **teoría de la decisión**, que combina la teoría de la probabilidad con la teoría de la utilidad, proporciona un marco completo y formal para la toma de decisiones (económicas o de otra índole) realizadas bajo incertidumbre, esto es, en casos en los que las descripciones probabilísticas capturan adecuadamente la forma en la que se toman las decisiones en el entorno; lo cual es adecuado para «grandes» economías en las que cada agente no necesita prestar atención a las acciones que lleven a cabo el resto de los agentes individualmente. Cuando se trata de «pequeñas» economías, la situación se asemeja más a la de un **juego**: las acciones de un jugador pueden afectar significativamente a la utilidad de otro (tanto positiva como negativamente). Los desarrollos de von Neumann y Morgenstern a partir de la **teoría de juegos** (*véase* también Luce y Raiffa, 1957) mostraban el hecho sorprendente de que, en algunos juegos, un agente racional debía actuar de forma aleatoria o, al menos, aleatoria en apariencia con respecto a sus contrincantes.

TEORÍA DE JUEGOS

La gran mayoría de los economistas no se preocuparon de la tercera cuestión mencionada anteriormente, es decir, cómo tomar decisiones racionales cuando los resultados de las acciones no son inmediatos y por el contrario se obtienen los resultados de las acciones de forma *secuencial*. El campo de la **investigación operativa** persigue este objetivo; dicho campo emergió en la Segunda Guerra Mundial con los esfuerzos llevados a cabo en el Reino Unido en la optimización de instalaciones de radar, y posteriormente en aplicaciones civiles relacionadas con la toma de decisiones de dirección complejas. El trabajo de Richard Bellman (1957) formaliza una clase de problemas de decisión secuencial llamados **procesos de decisión de Markov**, que se estudiarán en los Capítulos 17 y 21.

INVESTIGACIÓN OPERATIVA

El trabajo en la economía y la investigación operativa ha contribuido en gran medida a la noción de agente racional que aquí se presenta, aunque durante muchos años la investigación en el campo de la IA se ha desarrollado por sendas separadas. Una razón fue la **complejidad** aparente que trae consigo el tomar decisiones racionales. Herbert Simon (1916-2001), uno de los primeros en investigar en el campo de la IA, ganó el premio Nobel en Economía en 1978 por su temprano trabajo, en el que mostró que los modelos basados en **satisfacción** (que toman decisiones que son «suficientemente buenas», en vez de realizar cálculos laboriosos para alcanzar decisiones óptimas) proporcionaban una descripción mejor del comportamiento humano real (Simon, 1947). En los años 90, hubo un resurgimiento del interés en las técnicas de decisión teórica para sistemas basados en agentes (Wellman, 1995).

SATISFACCIÓN

Neurociencia (desde el año 1861 hasta el presente)

- ¿Cómo procesa información el cerebro?

NEUROCIENCIA

La **Neurociencia** es el estudio del sistema neurológico, y en especial del cerebro. La forma exacta en la que en un cerebro se genera el pensamiento es uno de los grandes misterios de la ciencia. Se ha observado durante miles de años que el cerebro está de alguna manera involucrado en los procesos de pensamiento, ya que fuertes golpes en la cabeza pueden ocasionar minusvalía mental. También es ampliamente conocido que los cerebros humanos son de alguna manera diferentes; aproximadamente en el 335 a.C. Aristóteles escribió, «de entre todos los animales el hombre tiene el cerebro más grande en proporción a su tamaño»[6]. Aunque, no fue hasta mediados del siglo XVIII cuando se aceptó mayoritariamente que el cerebro es la base de la conciencia. Hasta este momento, se pensaba que estaba localizado en el corazón, el bazo y la glándula pineal.

El estudio de Paul Broca (1824-1880) sobre la afasia (dificultad para hablar) en pacientes con el cerebro dañado, en 1861, le dio fuerza a este campo y convenció a la sociedad médica de la existencia de áreas localizadas en el cerebro responsables de funciones cognitivas específicas. En particular, mostró que la producción del habla se localizaba en una parte del hemisferio izquierdo; hoy en día conocida como el área de Broca[7]. En esta época ya se sabía que el cerebro estaba formado por células nerviosas o **neuronas**, pero no fue hasta 1873 cuando Camillo Golgi (1843-1926) desarrolló una técnica de coloración que permitió la observación de neuronas individuales en el cerebro (*véase* la Figura 1.2). Santiago Ramón y Cajal (1852-1934) utilizó esta técnica es sus estudios pioneros sobre la estructura neuronal del cerebro[8].

NEURONAS

En la actualidad se dispone de información sobre la relación existente entre las áreas del cerebro y las partes del cuerpo humano que controlan o de las que reciben impulsos

[6] Desde entonces, se ha descubierto que algunas especies de delfines y ballenas tienen cerebros relativamente grandes. Ahora se piensa que el gran tamaño de los cerebros humanos se debe en parte a la mejora reciente en su sistema de refrigeración.

[7] Muchos citan a Alexander Hood (1824) como una fuente posiblemente anterior.

[8] Golgi insistió en la creencia de que las funciones cerebrales se desarrollaron inicialmente en el medio continuo en el que las neuronas estaban inmersas, mientras que Cajal propuso la «doctrina neuronal». Ambos compartieron el premio Nobel en 1906 pronunciando un discurso de aceptación antagónico.

Figura 1.2 Partes de una célula nerviosa o neurona. Cada neurona contiene un cuerpo celular, o soma, que tiene un núcleo celular. Un número de fibras llamadas dendritas se ramifican a partir del cuerpo de la célula junto con una única fibra larga llamada axón. El axón se alarga considerablemente, mucho más que lo que se representa en esta imagen. Normalmente miden un centímetro (100 veces más que el diámetro del cuerpo de la célula), pero pueden alcanzar hasta un metro de longitud. Una neurona se conecta con entre 10 y 100.000 neuronas formando una maraña llamada sinapsis. Las señales se propagan de neurona a neurona mediante una reacción electroquímica complicada. Las señales controlan la actividad del cerebro a corto plazo, y permiten establecer cambios en la posición y conectividad de las neuronas a largo plazo. Se piensa que estos mecanismos forman la base del aprendizaje del cerebro. La mayoría del procesamiento de información se lleva a cabo en la corteza del cerebro, la capa más externa de éste. La unidad de organización básica es una columna de tejido de aproximadamente 0,5 mm de diámetro, con lo cual se amplía la profundidad total de la corteza cerebral, que en el caso de los humanos es de cuatro mm. Una columna contiene aproximadamente 20.000 neuronas.

sensoriales. Tales relaciones pueden cambiar de forma radical incluso en pocas semanas, y algunos animales parecen disponer de múltiples posibilidades. Más aún, no se tiene totalmente claro cómo algunas áreas se pueden encargar de ciertas funciones que eran responsabilidad de áreas dañadas. No hay prácticamente ninguna teoría que explique cómo se almacenan recuerdos individuales.

Los estudios sobre la actividad de los cerebros intactos comenzó en 1929 con el descubrimiento del electroencefalograma (EEG) desarrollado por Hans Berger. El reciente descubrimiento de las imágenes de resonancia magnética funcional (IRMF) (Ogawa *et al*., 1990) está proporcionando a los neurólogos imágenes detalladas de la actividad cerebral sin precedentes, permitiéndoles obtener medidas que se corresponden con procesos cognitivos en desarrollo de manera muy interesante. Este campo está evolucionando gracias a los avances en los estudios en celdas individuales y su actividad neuronal. A pesar de estos avances, nos queda un largo camino para llegar a comprender cómo funcionan todos estos procesos cognitivos.

La conclusión verdaderamente increíble es que *una colección de simples células puede llegar a generar razonamiento, acción, y conciencia* o, dicho en otras palabras, *los cerebros generan las inteligencias* (Searle, 1992). La única teoría alternativa es el misticismo: que nos dice que existe alguna esfera mística en la que las mentes operan fuera del control de la ciencia física.

Cerebros y computadores digitales realizan tareas bastante diferentes y tienen propiedades distintas. La Figura 1.3 muestra cómo hay 1.000 veces más neuronas en un cerebro humano medio que puertas lógicas en la UCP de un computador estándar. La ley de Moore[9] predice que el número de puertas lógicas de la UCP se igualará con el de neuronas del cerebro alrededor del año 2020. Por supuesto, poco se puede inferir de esta predicción; más aún, la diferencia en la capacidad de almacenamiento es insignificante comparada con las diferencias en la velocidad de intercambio y en paralelismo. Los circuitos de los computadores pueden ejecutar una instrucción en un nanosegundo, mientras que las neuronas son millones de veces más lentas. Las neuronas y las sinapsis del cerebro están activas simultáneamente, mientras que los computadores actuales tienen una o como mucho varias UCP. Por tanto, incluso sabiendo que un computador es un millón de veces más rápido en cuanto a su velocidad de intercambio, el cerebro acaba siendo 100.000 veces más rápido en lo que hace.

Psicología (desde el año 1879 hasta el presente)

- ¿Cómo piensan y actúan los humanos y los animales?

La psicología científica se inició con los trabajos del físico alemán Hermann von Helmholtz (1821-1894), según se referencia habitualmente, y su discípulo Wilhelm Wundt (1832-1920). Helmholtz aplicó el método científico al estudio de la vista humana, y su obra *Handbook of Physiological Optics*, todavía en nuestros días, se considera como «el tratado actual más importante sobre la física y la fisiología de la vista humana» (Nalwa, 1993, p. 15). En 1879, Wundt abrió el primer laboratorio de psicología experimental en

	Computador	Cerebro Humano
Unidades computacionales	1 UCP, 10^8 puertas	10^{11} neuronas
Unidades de Almacenamiento	10^{10} bits RAM	10^{11} neuronas
	10^{11} bits disco	10^{14} sinapsis
Duración de un ciclo	10^{-9} sec	10^{-3} sec
Ancho de banda	10^{10} bits/sec	10^{14} bits/sec
Memoria actualización/sec	10^9	10^{14}

Figura 1.3 Comparación básica entre los recursos de cómputo generales de que disponen los computadores (*circa 2003*) y el cerebro. Las cifras correspondientes a los computadores se han incrementado en al menos un factor 10 desde la primera edición de este libro, y se espera que suceda la mismo en esta década. Las cifras correspondientes al cerebro no han cambiado en los últimos 10.000 años.

[9] La ley de Moore dice que el número de transistores por pulgada cuadrada se duplica cada año o año y medio. La capacidad del cerebro humano se dobla aproximadamente cada dos o cuatro millones de años.

la Universidad de Leipzig. Wundt puso mucho énfasis en la realización de experimentos controlados cuidadosamente en la que sus operarios realizaban tareas de percepción o asociación al tiempo que sometían a introspección sus procesos mentales. Los meticulosos controles evolucionaron durante un largo período de tiempo hasta convertir la psicología en una ciencia, pero la naturaleza subjetiva de los datos hizo poco probable que un investigador pudiera contradecir sus propias teorías. Biólogos, estudiando el comportamiento humano, por el contrario, carecían de datos introspectivos y desarrollaron una metodología objetiva, tal y como describe H. S. Jennings (1906) en su influyente trabajo *Behavior of the Lower Organisms*. El movimiento **conductista**, liderado por John Watson (1878-1958) aplicó este punto de vista a los humanos, rechazando *cualquier* teoría en la que intervinieran procesos mentales, argumentando que la introspección no aportaba una evidencia fiable. Los conductistas insistieron en el estudio exclusivo de mediciones objetivas de percepciones (o *estímulos*) sobre animales y de las acciones resultantes (o *respuestas*). Construcciones mentales como conocimientos, creencias, objetivos y pasos en un razonamiento quedaron descartadas por ser consideradas «psicología popular» no científica. El conductismo hizo muchos descubrimientos utilizando ratas y palomas, pero tuvo menos éxito en la comprensión de los seres humanos. Aún así, su influencia en la psicología fue notable (especialmente en Estados Unidos) desde aproximadamente 1920 hasta 1960.

CONDUCTISMO

La conceptualización del cerebro como un dispositivo de procesamiento de información, característica principal de la **psicología cognitiva**, se remonta por lo menos a las obras de William James[10] (1842-1910). Helmholtz también pone énfasis en que la percepción entraña cierto tipo de inferencia lógica inconsciente. Este punto de vista cognitivo se vio eclipsado por el conductismo en Estados Unidos, pero en la Unidad de Psicología Aplicada de Cambridge, dirigida por Frederic Bartlett (1886-1969), los modelos cognitivos emergieron con fuerza. La obra *The Nature of Explanation*, de Kenneth Craik (1943), discípulo y sucesor de Bartlett, reestablece enérgicamente la legitimidad de términos «mentales» como creencias y objetivos, argumentando que son tan científicos como lo pueden ser la presión y la temperatura cuando se habla acerca de los gases, a pesar de que éstos estén formados por moléculas que no tienen ni presión ni temperatura. Craik establece tres elementos clave que hay que tener en cuenta para diseñar un agente basado en conocimiento: (1) el estímulo deberá ser traducido a una representación interna, (2) esta representación se debe manipular mediante procesos cognitivos para así generar nuevas representaciones internas, y (3) éstas, a su vez, se traducirán de nuevo en acciones. Dejó muy claro por qué consideraba que estos eran los requisitos idóneos para diseñar un agente:

PSICOLOGÍA COGNITIVA

> Si el organismo tiene en su cabeza «un modelo a pequeña escala» de la realidad externa y de todas sus posibles acciones, será capaz de probar diversas opciones, decidir cuál es la mejor, planificar su reacción ante posibles situaciones futuras antes de que éstas surjan, emplear lo aprendido de experiencias pasadas en situaciones presentes y futuras, y en todo momento, reaccionar ante los imprevistos que acontezcan de manera satisfactoria, segura y más competente (Craik, 1943).

[10] William James era hermano del novelista Henry James. Se comenta que Henry escribió novelas narrativas como si se tratara de psicología y William escribió sobre psicología como si se tratara de novelas narrativas.

Después de la muerte de Craik en un accidente de bicicleta en 1945, Donald Boadbent continuó su trabajo, y su libro *Perception and Communication* (1958) incluyó algunos de los primeros modelos de procesamiento de información del fenómeno psicológico. Mientras tanto, en Estados Unidos el desarrollo del modelo computacional llevo a la creación del campo de la **ciencia cognitiva**. Se puede decir que este campo comenzó en un simposio celebrado en el MIT, en septiembre de 1956 (como se verá a continuación este evento tuvo lugar sólo dos meses después de la conferencia en la que «nació» la IA). En este simposio, George Miller presentó *The Magic Number Seven*, Noam Chomsky presentó *Three Models of Language*, y Allen Newell y Herbert Simon presentaron *The Logic Theory Machine*. Estos tres artículos influyentes mostraron cómo se podían utilizar los modelos informáticos para modelar la psicología de la memoria, el lenguaje y el pensamiento lógico, respectivamente. Los psicólogos comparten en la actualidad el punto de vista común de que «la teoría cognitiva debe ser como un programa de computador» (Anderson, 1980), o dicho de otra forma, debe describir un mecanismo de procesamiento de información detallado, lo cual lleva consigo la implementación de algunas funciones cognitivas.

CIENCIA COGNITIVA

Ingeniería computacional (desde el año 1940 hasta el presente)

- ¿Cómo se puede construir un computador eficiente?

Para que la inteligencia artificial pueda llegar a ser una realidad se necesitan dos cosas: inteligencia y un artefacto. El computador ha sido el artefacto elegido. El computador electrónico digital moderno se inventó de manera independiente y casi simultánea por científicos en tres países involucrados en la Segunda Guerra Mundial. El equipo de Alan Turing construyó, en 1940, el primer computador *operacional* de carácter electromecánico, llamado Heath Robinson[11], con un único propósito: descifrar mensajes alemanes. En 1943 el mismo grupo desarrolló el Colossus, una máquina potente de propósito general basada en válvulas de vacío[12]. El primer computador operacional *programable* fue el Z-3, inventado por Konrad Zuse en Alemania, en 1941. Zuse también inventó los números de coma flotante y el primer lenguaje de programación de alto nivel, Plankalkül. El primer computador *electrónico*, el ABC, fue creado por John Atanasoff junto a su discípulo Clifford Berry entre 1940 y 1942 en la Universidad Estatal de Iowa. Las investigaciones de Atanasoff recibieron poco apoyo y reconocimiento; el ENIAC, desarrollado en el marco de un proyecto militar secreto, en la Universidad de Pensilvania, por un equipo en el que trabajaban entre otros John Mauchly y John Eckert, puede considerarse como el precursor de los computadores modernos.

Desde mediados del siglo pasado, cada generación de dispositivos *hardware* ha conllevado un aumento en la velocidad de proceso y en la capacidad de almacenamiento,

[11] Heath Robinson fue un caricaturista famoso por sus dibujos, que representaban artefactos de uso diario, caprichosos y absurdamente complicados, por ejemplo, uno para untar mantequilla en el pan tostado.

[12] En la postguerra, Turing quiso utilizar estos computadores para investigar en el campo de la IA, por ejemplo, desarrollando uno de los primeros programas para jugar a la ajedrez (Turing *et al*., 1953). El gobierno británico bloqueó sus esfuerzos.

así como una reducción de precios. La potencia de los computadores se dobla cada 18 meses aproximadamente y seguirá a este ritmo durante una o dos décadas más. Después, se necesitará ingeniería molecular y otras tecnologías novedosas.

Por supuesto que antes de la aparición de los computadores ya había dispositivos de cálculo. Las primeras máquinas automáticas, que datan del siglo XVII, ya se mencionaron en la página seis. La primera máquina programable fue un telar, desarrollado en 1805 por Joseph Marie Jacquard (1752-1834) que utilizaba tarjetas perforadas para almacenar información sobre los patrones de los bordados. A mediados del siglo XIX, Charles Babbage (1792-1871) diseñó dos máquinas, que no llegó a construir. La «Máquina de Diferencias», que aparece en la portada de este manuscrito, se concibió con la intención de facilitar los cálculos de tablas matemáticas para proyectos científicos y de ingeniería. Finalmente se construyó y se presentó en 1991 en el Museo de la Ciencia de Londres (Swade, 1993). La «Máquina Analítica» de Babbage era mucho más ambiciosa: incluía memoria direccionable, programas almacenados y saltos condicionales; fue el primer artefacto dotado de los elementos necesarios para realizar una computación universal. Ada Lovelace, colega de Babbage e hija del poeta Lord Byron, fue seguramente la primera programadora (el lenguaje de programación Ada se llama así en honor a esta programadora). Ella escribió programas para la inacabada Máquina Analítica e incluso especuló acerca de la posibilidad de que la máquina jugara al ajedrez y compusiese música.

La IA también tiene una deuda con la parte *software* de la informática que ha proporcionado los sistemas operativos, los lenguajes de programación, y las herramientas necesarias para escribir programas modernos (y artículos sobre ellos). Sin embargo, en este área la deuda se ha saldado: la investigación en IA ha generado numerosas ideas novedosas de las que se ha beneficiado la informática en general, como por ejemplo el tiempo compartido, los intérpretes imperativos, los computadores personales con interfaces gráficas y ratones, entornos de desarrollo rápido, listas enlazadas, administración automática de memoria, y conceptos claves de la programación simbólica, funcional, dinámica y orientada a objetos.

Teoría de control y cibernética (desde el año 1948 hasta el presente)

- ¿Cómo pueden los artefactos operar bajo su propio control?

Ktesibios de Alejandría (250 a.C.) construyó la primera máquina auto controlada: un reloj de agua con un regulador que mantenía el flujo de agua circulando por él, con un ritmo constante y predecible. Esta invención cambió la definición de lo que un artefacto podía hacer. Anteriormente, solamente seres vivos podían modificar su comportamiento como respuesta a cambios en su entorno. Otros ejemplos de sistemas de control auto regulables y retroalimentados son el motor de vapor, creado por James Watt (1736-1819), y el termostato, inventado por Cornelis Drebbel (1572-1633), que también inventó el submarino. La teoría matemática de los sistemas con retroalimentación estables se desarrolló en el siglo XIX.

TEORÍA DE CONTROL

La figura central del desarrollo de lo que ahora se llama la **teoría de control** fue Norbert Wiener (1894-1964). Wiener fue un matemático brillante que trabajó en sistemas de control biológicos y mecánicos y en sus vínculos con la cognición. De la misma forma que Craik (quien también utilizó sistemas de control como modelos psicológicos), Wiener y sus colegas Arturo Rosenblueth y Julian Bigelow desafiaron la ortodoxia conductista (Rosenblueth *et al*., 1943). Ellos veían el comportamiento determinista como algo emergente de un mecanismo regulador que intenta minimizar el «error» (la diferencia entre el estado presente y el estado objetivo). A finales de los años 40, Wiener, junto a Warren McCulloch, Walter Pitts y John von Neumann, organizaron una serie de conferencias en las que se exploraban los nuevos modelos cognitivos matemáticos y computacionales, e influyeron en muchos otros investigadores en el campo de las ciencias del comportamiento. El libro de Wiener, *Cybernetics* (1948), fue un *bestseller* y desveló al público las posibilidades de las máquinas con inteligencia artificial.

CIBERNÉTICA

FUNCIÓN OBJETIVO

La teoría de control moderna, especialmente la rama conocida como control óptimo estocástico, tiene como objetivo el diseño de sistemas que maximizan una **función objetivo** en el tiempo. Lo cual se asemeja ligeramente a nuestra visión de lo que es la IA: diseño de sistemas que se comportan de forma óptima. ¿Por qué, entonces, IA y teoría de control son dos campos diferentes, especialmente teniendo en cuenta la cercana relación entre sus creadores? La respuesta está en el gran acoplamiento existente entre las técnicas matemáticas con las que estaban familiarizados los investigadores y entre los conjuntos de problemas que se abordaban desde cada uno de los puntos de vista. El cálculo y el álgebra matricial, herramientas de la teoría de control, se utilizaron en la definición de sistemas que se podían describir mediante conjuntos fijos de variables continuas; más aún, el análisis exacto es sólo posible en sistemas *lineales*. La IA se fundó en parte para escapar de las limitaciones matemáticas de la teoría de control en los años 50. Las herramientas de inferencia lógica y computación permitieron a los investigadores de IA afrontar problemas relacionados con el lenguaje, visión y planificación, que estaban completamente fuera del punto de mira de la teoría de control.

Lingüística (desde el año 1957 hasta el presente)

- ¿Cómo está relacionado el lenguaje con el pensamiento?

En 1957, B. F. Skinner publicó *Verbal Behavior*. La obra presentaba una visión extensa y detallada desde el enfoque conductista al aprendizaje del lenguaje, y estaba escrita por los expertos más destacados de este campo. Curiosamente, una revisión de este libro llegó a ser tan famosa como la obra misma, y provocó el casi total desinterés por el conductismo. El autor de la revisión fue Noam Chomsky, quien acababa de publicar un libro sobre su propia teoría, *Syntactic Structures*. Chomsky mostró cómo la teoría conductista no abordaba el tema de la creatividad en el lenguaje: no explicaba cómo es posible que un niño sea capaz de entender y construir oraciones que nunca antes ha escuchado. La teoría de Chomsky (basada en modelos sintácticos que se remontaban al lingüista indio Panini, aproximadamente 350 a.C.) sí podía explicar lo anterior y, a diferencia de teorías anteriores, poseía el formalismo suficiente como para permitir su programación.

LINGÜÍSTICA COMPUTACIONAL

La lingüística moderna y la IA «nacieron», al mismo tiempo y maduraron juntas, solapándose en un campo híbrido llamado **lingüística computacional** o **procesamiento del lenguaje natural**. El problema del entendimiento del lenguaje se mostró pronto mucho más complejo de lo que se había pensado en 1957. El entendimiento del lenguaje requiere la comprensión de la materia bajo estudio y de su contexto, y no solamente el entendimiento de la estructura de las sentencias. Lo cual puede parecer obvio, pero no lo fue para la mayoría de la comunidad investigadora hasta los años 60. Gran parte de los primeros trabajos de investigación en el área de la **representación del conocimiento** (el estudio de cómo representar el conocimiento de forma que el computador pueda razonar a partir de dicha representación) estaban vinculados al lenguaje y a la búsqueda de información en el campo del lenguaje, y su base eran las investigaciones realizadas durante décadas en el análisis filosófico del lenguaje.

1.3 Historia de la inteligencia artificial

Una vez revisado el material básico estamos ya en condiciones de cubrir el desarrollo de la IA propiamente dicha.

Génesis de la inteligencia artificial (1943-1955)

Warren McCulloch y Walter Pitts (1943) han sido reconocidos como los autores del primer trabajo de IA. Partieron de tres fuentes: conocimientos sobre la fisiología básica y funcionamiento de las neuronas en el cerebro, el análisis formal de la lógica proposicional de Russell y Whitehead y la teoría de la computación de Turing. Propusieron un modelo constituido por neuronas artificiales, en el que cada una de ellas se caracterizaba por estar «activada» o «desactivada»; la «activación» se daba como respuesta a la estimulación producida por una cantidad suficiente de neuronas vecinas. El estado de una neurona se veía como «equivalente, de hecho, a una proposición con unos estímulos adecuados». Mostraron, por ejemplo, que cualquier función de cómputo podría calcularse mediante alguna red de neuronas interconectadas, y que todos los conectores lógicos (*and*, *or*, *not*, etc.) se podrían implementar utilizando estructuras de red sencillas. McCulloch y Pitts también sugirieron que redes adecuadamente definidas podrían aprender. Donald Hebb (1949) propuso y demostró una sencilla regla de actualización para modificar las intensidades de las conexiones entre neuronas. Su regla, ahora llamada **de aprendizaje Hebbiano o de Hebb**, sigue vigente en la actualidad.

Dos estudiantes graduados en el Departamento de Matemáticas de Princeton, Marvin Minsky y Dean Edmonds, construyeron el primer computador a partir de una red neuronal en 1951. El SNARC, como se llamó, utilizaba 3.000 válvulas de vacío y un mecanismo de piloto automático obtenido de los desechos de un avión bombardero B-24 para simular una red con 40 neuronas. El comité encargado de evaluar el doctorado de Minsky veía con escepticismo el que este tipo de trabajo pudiera considerarse como matemático, pero se dice que von Newmann dijo, «Si no lo es actualmente, algún día lo será».

Minsky posteriormente probó teoremas influyentes que mostraron las limitaciones de la investigación con redes neuronales.

Hay un número de trabajos iniciales que se pueden caracterizar como de IA, pero fue Alan Turing quien articuló primero una visión de la IA en su artículo *Computing Machinery and Intelligence*, en 1950. Ahí, introdujo la prueba de Turing, el aprendizaje automático, los algoritmos genéricos y el aprendizaje por refuerzo.

Nacimiento de la inteligencia artificial (1956)

Princeton acogió a otras de la figuras señeras de la IA, John McCarthy. Posteriormente a su graduación, McCarthy se transladó al Dartmouth College, que se erigiría en el lugar del nacimiento oficial de este campo. McCarthy convenció a Minsky, Claude Shannon y Nathaniel Rochester para que le ayudaran a aumentar el interés de los investigadores americanos en la teoría de autómatas, las redes neuronales y el estudio de la inteligencia. Organizaron un taller con una duración de dos meses en Darmouth en el verano de 1956. Hubo diez asistentes en total, entre los que se incluían Trenchard More de Princeton, Arthur Samuel de IBM, y Ray Solomonoff y Oliver Selfridge del MIT.

Dos investigadores del Carnegie Tech[13], Allen Newell y Herbert Simon, acapararon la atención. Si bien los demás también tenían algunas ideas y, en algunos casos, programas para aplicaciones determinadas como el juego de damas, Newell y Simon contaban ya con un programa de razonamiento, el Teórico Lógico (TL), del que Simon afirmaba: «Hemos inventado un programa de computación capaz de pensar de manera no numérica, con lo que ha quedado resuelto el venerable problema de la dualidad mente-cuerpo»[14]. Poco después del término del taller, el programa ya era capaz de demostrar gran parte de los teoremas del Capítulo 2 de *Principia Matemática* de Russell y Whitehead. Se dice que Russell se manifestó complacido cuando Simon le mostró que la demostración de un teorema que el programa había generado era más corta que la que aparecía en *Principia*. Los editores de la revista *Journal of Symbolic Logic* resultaron menos impresionados y rechazaron un artículo cuyos autores eran Newell, Simon y el Teórico Lógico (TL).

El taller de Dartmouth no produjo ningún avance notable, pero puso en contacto a las figuras importantes de este campo. Durante los siguientes 20 años, el campo estuvo dominado por estos personajes, así como por sus estudiantes y colegas del MIT, CMU, Stanford e IBM. Quizá lo último que surgió del taller fue el consenso en adoptar el nuevo nombre propuesto por McCarthy para este campo: **Inteligencia Artificial**. Quizá «racionalidad computacional» hubiese sido más adecuado, pero «IA» se ha mantenido.

Revisando la propuesta del taller de Dartmouth (McCarthy *et al.*, 1955), se puede apreciar por qué fue necesario para la IA convertirse en un campo separado. ¿Por qué

[13] Actualmente Universidad Carnegie Mellon (UCM).

[14] Newell y Simon también desarrollaron un lenguaje de procesamiento de listas, IPL, para poder escribir el TL. No disponían de un compilador y lo tradujeron a código máquina a mano. Para evitar errores, trabajaron en paralelo, diciendo en voz alta números binarios, conforme escribían cada instrucción para asegurarse de que ambos coincidían.

no todo el trabajo hecho en el campo de la IA se ha realizado bajo el nombre de teoría de control, o investigación operativa, o teoría de la decisión, que, después de todo, persiguen objetivos similares a los de la IA? O, ¿por qué no es la IA una rama de las matemáticas? La primera respuesta es que la IA desde el primer momento abarcó la idea de duplicar facultades humanas como la creatividad, la auto-mejora y el uso del lenguaje. Ninguno de los otros campos tenían en cuenta esos temas. La segunda respuesta está relacionada con la metodología. La IA es el único de estos campos que es claramente una rama de la informática (aunque la investigación operativa comparte el énfasis en la simulación por computador), además la IA es el único campo que persigue la construcción de máquinas que funcionen automáticamente en medios complejos y cambiantes.

Entusiasmo inicial, grandes esperanzas (1952-1969)

Los primeros años de la IA estuvieron llenos de éxitos (aunque con ciertas limitaciones). Teniendo en cuenta lo primitivo de los computadores y las herramientas de programación de aquella época, y el hecho de que sólo unos pocos años antes, a los computadores se les consideraba como artefactos que podían realizar trabajos aritméticos y nada más, resultó sorprendente que un computador hiciese algo remotamente inteligente. La comunidad científica, en su mayoría, prefirió creer que «una máquina nunca podría hacer *tareas*» (*véase* el Capítulo 26 donde aparece una extensa lista de *tareas* recopilada por Turing). Naturalmente, los investigadores de IA responderían demostrando la realización de una *tarea* tras otra. John McCarthy se refiere a esta época como la era de «¡Mira, mamá, ahora sin manos!».

Al temprano éxito de Newell y Simon siguió el del sistema de resolución general de problemas, o SRGP. A diferencia del Teórico Lógico, desde un principio este programa se diseñó para que imitara protocolos de resolución de problemas de los seres humanos. Dentro del limitado número de puzles que podía manejar, resultó que la secuencia en la que el programa consideraba que los subobjetivos y las posibles acciones eran semejantes a la manera en que los seres humanos abordaban los mismos problemas. Es decir, el SRGP posiblemente fue el primer programa que incorporó el enfoque de «pensar como un ser humano». El éxito del SRGP y de los programas que le siguieron, como los modelos de cognición, llevaron a Newell y Simon (1976) a formular la famosa hipótesis del **sistema de símbolos físicos**, que afirma que «un sistema de símbolos físicos tiene los medios suficientes y necesarios para generar una acción inteligente». Lo que ellos querían decir es que cualquier sistema (humano o máquina) que exhibiese inteligencia debería operar manipulando estructuras de datos compuestas por símbolos. Posteriormente se verá que esta hipótesis se ha modificado atendiendo a distintos puntos de vista.

SISTEMA DE SÍMBOLOS FÍSICOS

En IBM, Nathaniel Rochester y sus colegas desarrollaron algunos de los primeros programas de IA. Herbert Gelernter (1959) construyó el demostrador de teoremas de geometría (DTG), el cual era capaz de probar teoremas que muchos estudiantes de matemáticas podían encontrar muy complejos de resolver. A comienzos 1952, Arthur Samuel escribió una serie de programas para el juego de las damas que eventualmente aprendieron a jugar hasta alcanzar un nivel equivalente al de un *amateur*. De paso, echó por

tierra la idea de que los computadores sólo pueden hacer lo que se les dice: su programa pronto aprendió a jugar mejor que su creador. El programa se presentó en la televisión en febrero de 1956 y causó una gran impresión. Como Turing, Samuel tenía dificultades para obtener el tiempo de cómputo. Trabajaba por las noches y utilizaba máquinas que aún estaban en período de prueba en la planta de fabricación de IBM. El Capítulo 6 trata el tema de los juegos, y en el Capítulo 21 se describe con detalle las técnicas de aprendizaje utilizadas por Samuel.

John McCarthy se trasladó de Darmouth al MIT, donde realizó tres contribuciones cruciales en un año histórico: 1958. En el Laboratorio de IA del MIT Memo Número 1, McCarthy definió el lenguaje de alto nivel **Lisp**, que se convertiría en el lenguaje de programación dominante en la IA. Lisp es el segundo lenguaje de programación más antiguo que se utiliza en la actualidad, ya que apareció un año después de FORTRAN. Con Lisp, McCarthy tenía la herramienta que necesitaba, pero el acceso a los escasos y costosos recursos de cómputo aún era un problema serio. Para solucionarlo, él, junto a otros miembros del MIT, inventaron el tiempo compartido. También, en 1958, McCarthy publicó un artículo titulado *Programs with Common Sense*, en el que describía el Generador de Consejos, un programa hipotético que podría considerarse como el primer sistema de IA completo. Al igual que el Teórico Lógico y el Demostrador de Teoremas de Geometría, McCarthy diseñó su programa para buscar la solución a problemas utilizando el conocimiento. Pero, a diferencia de los otros, manejaba el conocimiento general del mundo. Por ejemplo, mostró cómo algunos axiomas sencillos permitían a un programa generar un plan para conducirnos hasta el aeropuerto y tomar un avión. El programa se diseñó para que aceptase nuevos axiomas durante el curso normal de operación, permitiéndole así ser competente en áreas nuevas, sin *necesidad de reprogramación*. El Generador de Consejos incorporaba así los principios centrales de la representación del conocimiento y el razonamiento: es útil contar con una representación formal y explícita del mundo y de la forma en que la acción de un agente afecta al mundo, así como, ser capaces de manipular estas representaciones con procesos deductivos. Es sorprendente constatar cómo mucho de lo propuesto en el artículo escrito en 1958 permanece vigente incluso en la actualidad.

1958 fue el año en el que Marvin Minsky se trasladó al MIT. Sin embargo, su colaboración inicial no duró demasiado. McCarthy se centró en la representación y el razonamiento con lógica formal, mientras que Minsky estaba más interesado en lograr que los programas funcionaran y eventualmente desarrolló un punto de vista anti-lógico. En 1963 McCarthy creó el Laboratorio de IA en Stanford. Su plan para construir la versión más reciente del Generador de Consejos con ayuda de la lógica sufrió un considerable impulso gracias al descubrimiento de J. A. Robinson del método de resolución (un algoritmo completo para la demostración de teoremas para la lógica de primer orden; *véase* el Capítulo 9). El trabajo realizado en Stanford hacía énfasis en los métodos de propósito general para el razonamiento lógico. Algunas aplicaciones de la lógica incluían los sistemas de planificación y respuesta a preguntas de Cordell Green (1969b), así como el proyecto de robótica de Shakey en el nuevo Instituto de Investigación de Stanford (Stanford Research Institute, SRI). Este último proyecto, comentado en detalle en el Capítulo 25, fue el primero que demostró la total integración del razonamiento lógico y la actividad física.

Minsky supervisó el trabajo de una serie de estudiantes que eligieron un número de problemas limitados cuya solución pareció requerir inteligencia. Estos dominios limi-

MICROMUNDOS

tados se conocen como **micromundos**. El programa SAINT de James Slagle (1963a) fue capaz de resolver problemas de integración de cálculo en forma cerrada, habituales en los primeros cursos de licenciatura. El programa ANALOGY de Tom Evans (1968) resolvía problemas de analogía geométrica que se aplicaban en las pruebas de medición de inteligencia, semejante al de la Figura 1.4. El programa STUDENT de Daniel Bobrow (1967) podía resolver problemas de álgebra del tipo:

> Si el número de clientes de Tom es dos veces el cuadrado del 20 por ciento de la cantidad de anuncios que realiza, y éstos ascienden a 45, ¿cuántos clientes tiene Tom?

El micromundo más famoso fue el mundo de los bloques, que consiste en un conjunto de bloques sólidos colocados sobre una mesa (más frecuentemente, en la simulación de ésta), como se muestra en la Figura 1.5. Una tarea típica de este mundo es la reordenación de los bloques de cierta manera, con la ayuda de la mano de un robot que es capaz de tomar un bloque cada vez. El mundo de los bloques fue el punto de partida para el proyecto de visión de David Huffman (1971), la visión y el trabajo de propagación con restricciones de David Waltz (1975), la teoría del aprendizaje de Patrick Winston (1970), del programa para la comprensión de lenguaje natural de Terry Winograd (1972) y del planificador de Scott Fahlman (1974).

El trabajo realizado por McCulloch y Pitts con redes neuronales hizo florecer esta área. El trabajo de Winograd y Cowan (1963) mostró cómo un gran número de elementos podría representar un concepto individual de forma colectiva, lo cual llevaba consigo un aumento proporcional en robustez y paralelismo. Los métodos de aprendizaje de Hebb se reforzaron con las aportaciones de Bernie Widrow (Widrow y Hoff, 1960; Widrow, 1962), quien llamó **adalines** a sus redes, y por Frank Rosenblatt (1962) con sus **perceptrones**. Rosenblatt demostró el famoso teorema del perceptrón, con lo que mostró que su algoritmo de aprendizaje podría ajustar las intensidades de las conexiones de un perceptrón para que se adaptaran a los datos de entrada, siempre y cuando existiera una correspondencia. Estos temas se explicarán en el Capítulo 20.

Figura 1.4 Ejemplo de un problema resuelto por el programa ANALOGY de Evans.

Figura 1.5 Esquema del mundo de los bloques. SHRDLU (Winograd, 1972) ha completado su tarea, «Busca un bloque que sea más alto que el que ahora tienes y ponlo en la caja».

Una dosis de realidad (1966-1973)

Desde el principio, los investigadores de IA hicieron públicas, sin timidez, predicciones sobre el éxito que les esperaba. Con frecuencia, se cita el siguiente comentario realizado por Herbert Simon en 1957:

> Sin afán de sorprenderlos y dejarlos atónitos, pero la forma más sencilla que tengo de resumirlo es diciéndoles que actualmente en el mundo existen máquinas capaces de pensar, aprender y crear. Además, su aptitud para hacer lo anterior aumentará rápidamente hasta que (en un futuro previsible) la magnitud de problemas que serán capaces de resolver irá a la par que la capacidad de la mente humana para hacer lo mismo.

Términos como «futuro previsible» pueden interpretarse de formas distintas, pero Simon también hizo predicciones más concretas: como que en diez años un computador llegaría a ser campeón de ajedrez, y que se podría demostrar un importante teorema matemático con una máquina. Estas predicciones se cumplirían (al menos en parte) dentro de los siguientes 40 años y no en diez. El exceso de confianza de Simon se debió a la prometedora actuación de los primeros sistemas de IA en problemas simples. En la mayor parte de los casos resultó que estos primeros sistemas fallaron estrepitosamente cuando se utilizaron en problemas más variados o de mayor dificultad.

El primer tipo de problemas surgió porque la mayoría de los primeros programas contaban con poco o ningún conocimiento de las materia objeto de estudio; obtenían resultados gracias a sencillas manipulaciones sintácticas. Una anécdota típica tuvo lugar cuando se comenzaba a trabajar en la traducción automática, actividad que recibía un

generoso patrocinio del Consejo Nacional para la Investigación de Estados Unidos en un intento de agilizar la traducción de artículos científicos rusos en vísperas del lanzamiento del Sputnik en 1957. Al principio se consideró que todo se reduciría a sencillas transformaciones sintácticas apoyadas en las gramáticas rusas e inglesa y al emplazamiento de palabras mediante un diccionario electrónico, lo que bastaría para obtener el significado exacto de las oraciones. La realidad es que para traducir es necesario contar con un conocimiento general sobre el tema, que permita resolver ambigüedades y así, precisar el contenido de una oración. La famosa retraducción del ruso al inglés de la frase «el espíritu es fuerte pero la carne es débil», cuyo resultado fue «el vodka es bueno pero la carne está podrida» es un buen ejemplo del tipo de dificultades que surgieron. En un informe presentado en 1966, el comité consultivo declaró que «no se ha logrado obtener ninguna traducción de textos científicos generales ni se prevé obtener ninguna en un futuro inmediato». Se canceló todo el patrocinio del gobierno estadounidense que se había asignado a los proyectos académicos sobre traducción. Hoy día, la traducción automática es una herramienta imperfecta pero de uso extendido en documentos técnicos, comerciales, gubernamentales y de Internet.

El segundo problema fue que muchos de los problemas que se estaban intentando resolver mediante la IA eran intratables. La mayoría de los primeros programas de IA resolvían problemas experimentando con diversos pasos hasta que se llegara a encontrar una solución. Esto funcionó en los primeros programas debido a que los micromundos con los que se trabajaba contenían muy pocos objetos, y por lo tanto muy pocas acciones posibles y secuencias de soluciones muy cortas. Antes de que se desarrollara la teoría de la complejidad computacional, se creía que para «aumentar» el tamaño de los programas de forma que estos pudiesen solucionar grandes problemas sería necesario incrementar la velocidad del *hardware* y aumentar las memorias. El optimismo que acompañó al logro de la demostración de problemas, por ejemplo, pronto se vio eclipsado cuando los investigadores fracasaron en la demostración de teoremas que implicaban más de unas pocas decenas de condiciones. *El hecho de que, en principio, un programa sea capaz de encontrar una solución no implica que tal programa encierre todos los mecanismos necesarios para encontrar la solución en la práctica.*

La ilusoria noción de una ilimitada capacidad de cómputo no sólo existió en los programas para la resolución de problemas. Los primeros experimentos en el campo de la **evolución automática** (ahora llamados **algoritmos genéticos**) (Friedberg, 1958; Friedberg *et al*., 1959) estaban basados en la, sin duda correcta, premisa de que efectuando una adecuada serie de pequeñas mutaciones a un programa de código máquina se podría generar un programa con buen rendimiento aplicable en cualquier tarea sencilla. Después surgió la idea de probar con mutaciones aleatorias aplicando un proceso de selección con el fin de conservar aquellas mutaciones que hubiesen demostrado ser más útiles. No obstante, las miles de horas de CPU dedicadas, no dieron lugar a ningún avance tangible. Los algoritmos genéticos actuales utilizan representaciones mejores y han tenido más éxito.

EVOLUCIÓN AUTOMÁTICA

La incapacidad para manejar la «explosión combinatoria» fue una de las principales críticas que se hicieron a la IA en el informe de Lighthill (Lighthill, 1973), informe en el que se basó la decisión del gobierno británico para retirar la ayuda a las investigaciones sobre IA, excepto en dos universidades. (La tradición oral presenta un cuadro un

poco distinto y más animado, en el que se vislumbran ambiciones políticas y animadversiones personales, cuya descripción está fuera del ámbito de esta obra.)

El tercer obstáculo se derivó de las limitaciones inherentes a las estructuras básicas que se utilizaban en la generación de la conducta inteligente. Por ejemplo, en 1969, en el libro de Minsky y Papert, *Perceptrons*, se demostró que si bien era posible lograr que los perceptrones (una red neuronal simple) aprendieran cualquier cosa que pudiesen representar, su capacidad de representación era muy limitada. En particular, un perceptrón con dos entradas no se podía entrenar para que aprendiese a reconocer cuándo sus dos entradas eran diferentes. Si bien los resultados que obtuvieron no eran aplicables a redes más complejas multicapa, los fondos para la investigación de las redes neuronales se redujeron a prácticamente nada. Es irónico que los nuevos algoritmos de aprendizaje de retroalimentación utilizados en las redes multicapa y que fueron la causa del gran resurgimiento de la investigación en redes neuronales de finales de los años 80, en realidad, se hayan descubierto por primera vez en 1969 (Bryson y Ho, 1969).

Sistemas basados en el conocimiento: ¿clave del poder? (1969-1979)

El cuadro que dibujaba la resolución de problemas durante la primera década de la investigación en la IA estaba centrado en el desarrollo de mecanismos de búsqueda de propósito general, en los que se entrelazaban elementos de razonamiento básicos para encontrar así soluciones completas. A estos procedimientos se les ha denominado **métodos débiles**, debido a que no tratan problemas más amplios o más complejos. La alternativa a los métodos débiles es el uso de conocimiento específico del dominio que facilita el desarrollo de etapas de razonamiento más largas, pudiéndose así resolver casos recurrentes en dominios de conocimiento restringido. Podría afirmarse que para resolver un problema en la práctica, es necesario saber de antemano la correspondiente respuesta.

MÉTODOS DÉBILES

El programa DENDRAL (Buchanan *et al*., 1969) constituye uno de los primeros ejemplos de este enfoque. Fue diseñado en Stanford, donde Ed Feigenbaum (discípulo de Herbert Simon), Bruce Buchanan (filósofo convertido en informático) y Joshua Lederberg (genetista ganador del Premio Nobel) colaboraron en la solución del problema de inferir una estructura molecular a partir de la información proporcionada por un espectrómetro de masas. El programa se alimentaba con la fórmula elemental de la molécula (por ejemplo, $C_6H_{13}NO_2$) y el espectro de masas, proporcionando las masas de los distintos fragmentos de la molécula generada después de ser bombardeada con un haz de electrones. Por ejemplo, un espectro de masas con un pico en $m = 15$, correspondería a la masa de un fragmento de metilo (CH_3).

La versión más simple del programa generaba todas las posibles estructuras que correspondieran a la fórmula, luego predecía el espectro de masas que se observaría en cada caso, y comparaba éstos con el espectro real. Como era de esperar, el método anterior resultó pronto inviable para el caso de moléculas con un tamaño considerable. Los creadores de DENDRAL consultaron con químicos analíticos y se dieron cuenta de que éstos trabajaban buscando patrones conocidos de picos en el espectro que sugerían estructuras comunes en la molécula. Por ejemplo, par identificar el subgrupo (con un peso de 28) de las cetonas (C=0) se empleó la siguiente regla:

si hay dos picos en x_1 y x_2 tales que
a) $x_1 + x_2 = M + 28$ (siendo M la masa de toda la molécula);
b) $x_1 - 28$ es un pico alto;
c) $x_2 - 28$ es un pico alto;
d) al menos una de x_1 y x_2 es alta.
entonces existe un subgrupo de cetonas

Al reconocer que la molécula contiene una subestructura concreta se reduce el número de posibles candidatos de forma considerable. La potencia de DENDRAL se basaba en que:

> Toda la información teórica necesaria para resolver estos problemas se ha proyectado desde su forma general [componente predicho por el espectro] («primeros principios») a formas eficientes especiales («recetas de cocina»). (Feigenbaum *et al*., 1971)

La trascendencia de DENDRAL se debió a ser el primer sistema de *conocimiento intenso* que tuvo éxito: su base de conocimiento estaba formada por grandes cantidades de reglas de propósito particular. En sistemas diseñados posteriormente se incorporaron también los elementos fundamentales de la propuesta de McCarthy para el Generador de Consejos, la nítida separación del conocimiento (en forma de reglas) de la parte correspondiente al razonamiento.

Teniendo en cuenta esta lección, Feigenbaum junto con otros investigadores de Stanford dieron comienzo al Proyecto de Programación Heurística, PPH, dedicado a determinar el grado con el que la nueva metodología de los **sistemas expertos** podía aplicarse a otras áreas de la actividad humana. El siguiente gran esfuerzo se realizó en el área del diagnóstico médico. Feigenbaum, Buchanan y el doctor Edward Shortliffe diseñaron el programa MYCIN, para el diagnóstico de infecciones sanguíneas. Con 450 reglas aproximadamente, MYCIN era capaz de hacer diagnósticos tan buenos como los de un experto y, desde luego, mejores que los de un médico recién graduado. Se distinguía de DENDRAL en dos aspectos principalmente. En primer lugar, a diferencia de las reglas de DENDRAL, no se contaba con un modelo teórico desde el cual se pudiesen deducir las reglas de MYCIN. Fue necesario obtenerlas a partir de extensas entrevistas con los expertos, quienes las habían obtenido de libros de texto, de otros expertos o de su experiencia directa en casos prácticos. En segundo lugar, las reglas deberían reflejar la incertidumbre inherente al conocimiento médico. MYCIN contaba con un elemento que facilitaba el cálculo de incertidumbre denominado **factores de certeza** (*véase* el Capítulo 13), que al parecer (en aquella época) correspondía muy bien a la manera como los médicos ponderaban las evidencias al hacer un diagnóstico.

SISTEMAS EXPERTOS

La importancia del conocimiento del dominio se demostró también en el área de la comprensión del lenguaje natural. Aunque el sistema SHRDLU de Winograd para la comprensión del lenguaje natural había suscitado mucho entusiasmo, su dependencia del análisis sintáctico provocó algunos de los mismos problemas que habían aparecido en los trabajos realizados en la traducción automática. Era capaz de resolver los problemas de ambigüedad e identificar los pronombres utilizados, gracias a que se había diseñado especialmente para un área (el mundo de los bloques). Fueron varios los investigadores que, como Eugene Charniak, estudiante de Winograd en el MIT, opinaron que para una sólida comprensión del lenguaje era necesario contar con un conocimiento general sobre el mundo y un método general para usar ese conocimiento.

En Yale, el lingüista transformado en informático Roger Schank reforzó lo anterior al afirmar: «No existe eso que llaman sintaxis», lo que irritó a muchos lingüistas, pero sirvió para iniciar un útil debate. Schank y sus estudiantes diseñaron una serie de programas (Schank y Abelson, 1977; Wilensky, 1978; Schank y Riesbeck, 1981; Dyer, 1983) cuyo objetivo era la comprensión del lenguaje natural. El foco de atención estaba menos en el lenguaje *per se* y más en los problemas vinculados a la representación y razonamiento del conocimiento necesario para la comprensión del lenguaje. Entre los problemas estaba el de la representación de situaciones estereotipo (Cullingford, 1981), la descripción de la organización de la memoria humana (Rieger, 1976; Kolodner, 1983) y la comprensión de planes y objetivos (Wilensky, 1983).

El crecimiento generalizado de aplicaciones para solucionar problemas del mundo real provocó el respectivo aumento en la demanda de esquemas de representación del conocimiento que funcionaran. Se desarrolló una considerable cantidad de lenguajes de representación y razonamiento diferentes. Algunos basados en la lógica, por ejemplo el lenguaje Prolog gozó de mucha aceptación en Europa, aceptación que en Estados Unidos fue para la familia del PLANNER. Otros, siguiendo la noción de **marcos** de Minsky (1975), se decidieron por un enfoque más estructurado, al recopilar información sobre objetos concretos y tipos de eventos, organizando estos tipos en grandes jerarquías taxonómicas, similares a las biológicas.

MARCOS

La IA se convierte en una industria (desde 1980 hasta el presente)

El primer sistema experto comercial que tuvo éxito, R1, inició su actividad en Digital Equipment Corporation (McDermott, 1982). El programa se utilizaba en la elaboración de pedidos de nuevos sistemas informáticos. En 1986 representaba para la compañía un ahorro estimado de 40 millones de dólares al año. En 1988, el grupo de Inteligencia Artificial de DEC había distribuido ya 40 sistemas expertos, y había más en camino. Du Pont utilizaba ya 100 y estaban en etapa de desarrollo 500 más, lo que le generaba ahorro de diez millones de dólares anuales aproximadamente. Casi todas las compañías importantes de Estados Unidos contaban con su propio grupo de IA, en el que se utilizaban o investigaban sistemas expertos.

En 1981 los japoneses anunciaron el proyecto «Quinta Generación», un plan de diez años para construir computadores inteligentes en los que pudiese ejecutarse Prolog. Como respuesta Estados Unidos constituyó la Microelectronics and Computer Technology Corporation (MCC), consorcio encargado de mantener la competitividad nacional en estas áreas. En ambos casos, la IA formaba parte de un gran proyecto que incluía el diseño de chips y la investigación de la relación hombre máquina. Sin embargo, los componentes de IA generados en el marco de MCC y del proyecto Quinta Generación nunca alcanzaron sus objetivos. En el Reino Unido, el informe Alvey restauró el patrocinio suspendido por el informe Lighthill[15].

[15] Para evitar confusiones, se creó un nuevo campo denominado Sistemas Inteligentes Basados en Conocimiento (IKBS, *Intelligent Knowledge-Based Systems*) ya que la Inteligencia Artificial había sido oficialmente cancelada.

En su conjunto, la industria de la IA creció rápidamente, pasando de unos pocos millones de dólares en 1980 a billones de dólares en 1988. Poco después de este período llegó la época llamada «El Invierno de la IA», que afectó a muchas empresas que no fueron capaces de desarrollar los extravagantes productos prometidos.

Regreso de las redes neuronales (desde 1986 hasta el presente)

Aunque la informática había abandonado de manera general el campo de las redes neuronales a finales de los años 70, el trabajo continuó en otros campos. Físicos como John Hopfield (1982) utilizaron técnicas de la mecánica estadística para analizar las propiedades de almacenamiento y optimización de las redes, tratando colecciones de nodos como colecciones de átomos. Psicólogos como David Rumelhart y Geoff Hinton continuaron con el estudio de modelos de memoria basados en redes neuronales. Como se verá en el Capítulo 20, el impulso más fuerte se produjo a mediados de la década de los 80, cuando por lo menos cuatro grupos distintos reinventaron el algoritmo de aprendizaje de retroalimentación, mencionado por vez primera en 1969 por Bryson y Ho. El algoritmo se aplicó a diversos problemas de aprendizaje en los campos de la informática y la psicología, y la gran difusión que conocieron los resultados obtenidos, publicados en la colección *Parallel Distributed Processing* (Rumelhart y McClelland, 1986), suscitó gran entusiasmo.

CONEXIONISTAS

Aquellos modelos de inteligencia artificial llamados **conexionistas**[16] fueron vistos por algunos como competidores tanto de los modelos simbólicos propuestos por Newell y Simon como de la aproximación lógica de McCarthy entre otros (Smolensky, 1988). Puede parecer obvio que los humanos manipulan símbolos hasta cierto nivel, de hecho, el libro *The Symbolic Species* (1997) de Terrence Deacon sugiere que esta es la *característica que define* a los humanos, pero los conexionistas más ardientes se preguntan si la manipulación de los símbolos desempeña algún papel justificable en determinados modelos de cognición. Este interrogante no ha sido aún clarificado, pero la tendencia actual es que las aproximaciones conexionistas y simbólicas son complementarias y no competidoras.

IA se convierte en una ciencia (desde 1987 hasta el presente)

En los últimos años se ha producido una revolución tanto en el contenido como en la metodología de trabajo en el campo de la inteligencia artificial.[17] Actualmente es más usual el desarrollo sobre teorías ya existentes que proponer teorías totalmente novedo-

[16] Se usa la traducción literal del término *connectionist* por no existir un término equivalente en español (*N. del RT*).

[17] Hay quien ha caracterizado este cambio como la victoria de los pulcros (aquellos que consideran que las teorías de IA deben basarse rigurosamente en las matemáticas) sobre los desaliñados (aquellos que después de intentar muchas ideas, escriben algunos programas y después evalúan las que aparentemente funcionan). Ambos enfoques son útiles. Esta tendencia en favor de una mayor pulcritud es señal de que el campo ha alcanzado cierto nivel de estabilidad y madurez. Lo cual no implica que tal estabilidad se puede ver alterada con el surgimiento de otras ideas poco estructuradas.

sas, tomar como base rigurosos teoremas o sólidas evidencias experimentales más que intuición, y demostrar la utilidad de las aplicaciones en el mundo real más que crear ejemplos de juguete.

La IA se fundó en parte en el marco de una rebelión en contra de las limitaciones de los campos existentes como la teoría de control o la estadística, y ahora abarca estos campos. Tal y como indica David McAllester (1998),

> En los primeros años de la IA parecía perfectamente posible que las nuevas formas de la computación simbólica, por ejemplo, los marcos y las redes semánticas, hicieran que la mayor parte de la teoría clásica pasara a ser obsoleta. Esto llevó a la IA a una especie de aislamiento, que la separó del resto de las ciencias de la computación. En la actualidad se está abandonando este aislamiento. Existe la creencia de que el aprendizaje automático no se debe separar de la teoría de la información, que el razonamiento incierto no se debe separar de los modelos estocásticos, de que la búsqueda no se debe aislar de la optimización clásica y el control, y de que el razonamiento automático no se debe separar de los métodos formales y del análisis estático.

En términos metodológicos, se puede decir, con rotundidad, que la IA ya forma parte del ámbito de los métodos científicos. Para que se acepten, las hipótesis se deben someter a rigurosos experimentos empíricos, y los resultados deben analizarse estadísticamente para identificar su relevancia (Cohen, 1995). El uso de Internet y el compartir repositorios de datos de prueba y código, ha hecho posible que ahora se puedan contrastar experimentos.

Un buen modelo de la tendencia actual es el campo del reconocimiento del habla. En la década de los 70 se sometió a prueba una gran variedad de arquitecturas y enfoques. Muchos de ellos fueron un tanto *ad hoc* y resultaban frágiles, y fueron probados sólo en unos pocos ejemplos elegidos especialmente. En años recientes, las aproximaciones basadas en los **modelos de Markov ocultos**, MMO, han pasado a dominar el área. Dos son las características de los MMO que tienen relevancia. Primero, se basan en una rigurosa teoría matemática, lo cual ha permitido a los investigadores del lenguaje basarse en los resultados de investigaciones matemáticas hechas en otros campos a lo largo de varias décadas. En segundo lugar, los modelos se han generado mediante un proceso de aprendizaje en grandes *corpus* de datos de lenguaje reales. Lo cual garantiza una funcionalidad robusta, y en sucesivas pruebas ciegas, los MMO han mejorado sus resultados a un ritmo constante. La tecnología del habla y el campo relacionado del reconocimiento de caracteres manuscritos están actualmente en transición hacia una generalizada utilización en aplicaciones industriales y de consumo.

Las redes neuronales también siguen esta tendencia. La mayor parte del trabajo realizado con redes neuronales en la década de los 80 se realizó con la idea de dejar a un lado lo que se podía hacer y de descubrir en qué se diferenciaban las redes neuronales de otras técnicas «tradicionales». La utilización de metodologías mejoradas y marcos teóricos, ha autorizado que este campo alcance un grado de conocimiento que ha permitido que ahora las redes neuronales se puedan comparar con otras técnicas similares de campos como la estadística, el reconocimiento de patrones y el aprendizaje automático, de forma que las técnicas más prometedoras pueden aplicarse a cualquier problema. Como resultado de estos desarrollos, la tecnología denominada **minería de datos** ha generado una nueva y vigorosa industria.

MINERÍA DE DATOS

La aparición de *Probabilistic Reasoning in Intelligent Systems* de Judea Pearl (1988) hizo que se aceptara de nuevo la probabilidad y la teoría de la decisión como parte de la IA, como consecuencia del resurgimiento del interés despertado y gracias especialmente al artículo *In Defense of Probability* de Peter Cheeseman (1985). El formalismo de las **redes de Bayes** apareció para facilitar la representación eficiente y el razonamiento riguroso en situaciones en las que se disponía de conocimiento incierto. Este enfoque supera con creces muchos de los problemas de los sistemas de razonamiento probabilístico de las décadas de los 60 y 70; y ahora domina la investigación de la IA en el razonamiento incierto y los sistemas expertos. Esta aproximación facilita el aprendizaje a partir de la experiencia, y combina lo mejor de la IA clásica y las redes neuronales. El trabajo de Judea Pearl (1982a) y de Eric Horvitz y David Heckerman (Horvitz y Heckerman, 1986; Horvitz *et al*., 1986) sirvió para promover la noción de sistemas expertos *normativos*: es decir, los que actúan racionalmente de acuerdo con las leyes de la teoría de la decisión, sin que intenten imitar las etapas de razonamiento de los expertos humanos. El sistema operativo Windows™ incluye varios sistemas expertos de diagnóstico normativos para la corrección de problemas. En los Capítulos 13 y 16 se aborda este tema.

Revoluciones similares y suaves se han dado en robótica, visión por computador, y aprendizaje automático. La comprensión mejor de los problemas y de su complejidad, junto a un incremento en la sofisticación de las matemáticas ha facilitado el desarrollo de una agenda de investigación y de métodos más robustos. En cualquier caso, la formalización y especialización ha llevado también a la fragmentación: áreas como la visión y la robótica están cada vez más aislados de la «rama central» de la IA. La concepción unificadora de IA como diseño de agentes racionales puede facilitar la unificación de estos campos diferentes.

Emergencia de los sistemas inteligentes (desde 1995 hasta el presente)

Quizás animados por el progreso en la resolución de subproblemas de IA, los investigadores han comenzado a trabajar de nuevo en el problema del «agente total». El trabajo de Allen Newell, John Laird, y Paul Rosenbloom en SOAR (Newell, 1990; Laird *et al*., 1987) es el ejemplo mejor conocido de una arquitectura de agente completa. El llamado «movimiento situado» intenta entender la forma de actuar de los agentes inmersos en entornos reales, que disponen de sensores de entradas continuas. Uno de los medios más importantes para los agentes inteligentes es Internet. Los sistemas de IA han llegado a ser tan comunes en aplicaciones desarrolladas para la Web que el sufijo «-bot» se ha introducido en el lenguaje común. Más aún, tecnologías de IA son la base de muchas herramientas para Internet, como por ejemplo motores de búsqueda, sistemas de recomendación, y los sistemas para la construcción de portales Web.

Además de la primera edición de este libro de texto (Russell y Norvig, 1995), otros libros de texto han adoptado recientemente la perspectiva de agentes (Poole *et al*., 1998; Nilsson, 1998). Una de las conclusiones que se han extraído al tratar de construir agentes completos ha sido que se deberían reorganizar los subcampos aislados de la IA para

que sus resultados se puedan interrelacionar. En particular, ahora se cree mayoritariamente que los sistemas sensoriales (visión, sónar, reconocimiento del habla, etc.) no pueden generar información totalmente fidedigna del medio en el que habitan. Otra segunda consecuencia importante, desde la perspectiva del agente, es que la IA se ha ido acercando a otros campos, como la teoría de control y la economía, que también tratan con agentes.

1.4 El estado del arte

¿Qué es capaz de hacer la IA hoy en día? Responder de manera concisa es difícil porque hay muchas actividades divididas en muchos subcampos. Aquí se presentan unas cuantas aplicaciones; otras aparecerán a lo largo del texto.

Planificación autónoma: a un centenar de millones de millas de la Tierra, el programa de la NASA Agente Remoto se convirtió en el primer programa de planificación autónoma a bordo que controlaba la planificación de las operaciones de una nave espacial desde abordo (Jonsson *et al*., 2000). El Agente Remoto generaba planes a partir de objetivos generales especificados desde tierra, y monitorizaba las operaciones de la nave espacial según se ejecutaban los planes (detección, diagnóstico y recuperación de problemas según ocurrían).

Juegos: Deep Blue de IBM fue el primer sistema que derrotó a un campeón mundial en una partida de ajedrez cuando superó a Garry Kasparov por un resultado de 3.5 a 2.5 en una partida de exhibición (Goodman y Keene, 1997). Kasparov dijo que había percibido un «nuevo tipo de inteligencia» al otro lado del tablero. La revista *Newsweek* describió la partida como «La partida final». El valor de las acciones de IBM se incrementó en 18 billones de dólares.

Control autónomo: el sistema de visión por computador ALVINN fue entrenado para dirigir un coche de forma que siguiese una línea. Se instaló en una furgoneta controlada por computador en el NAVLAB de UCM y se utilizó para dirigir al vehículo por Estados Unidos. Durante 2.850 millas controló la dirección del vehículo en el 98 por ciento del trayecto. Una persona lo sustituyó en el dos por ciento restante, principalmente en vías de salida. El NAVLAB posee videocámaras que transmiten imágenes de la carretera a ALVINN, que posteriormente calcula la mejor dirección a seguir, basándose en las experiencias acumuladas en los viajes de entrenamiento.

Diagnosis: los programas de diagnóstico médico basados en el análisis probabilista han llegado a alcanzar niveles similares a los de médicos expertos en algunas áreas de la medicina. Heckerman (1991) describe un caso en el que un destacado experto en la patología de los nodos linfáticos se mofó del diagnóstico generado por un programa en un caso especialmente difícil. El creador del programa le sugirió que le preguntase al computador cómo había generado el diagnóstico. La máquina indicó los factores más importantes en los que había basado su decisión y explicó la ligera interacción existente entre varios de los síntomas en este caso. Eventualmente, el experto aceptó el diagnóstico del programa.

Planificación logística: durante la crisis del Golfo Pérsico de 1991, las fuerzas de Estados Unidos desarrollaron la herramienta *Dynamic Analysis and Replanning Tool*

(DART) (Cross y Walker, 1994), para automatizar la planificación y organización logística del transporte. Lo que incluía hasta 50.000 vehículos, carga y personal a la vez, teniendo en cuenta puntos de partida, destinos, rutas y la resolución de conflictos entre otros parámetros. Las técnicas de planificación de IA permitieron que se generara un plan en cuestión de horas que podría haber llevado semanas con otros métodos. La agencia DARPA (*Defense Advanced Research Project Agency*) afirmó que esta aplicación por sí sola había más que amortizado los 30 años de inversión de DARPA en IA.

Robótica: muchos cirujanos utilizan hoy en día asistentes robot en operaciones de microcirugía. HipNav (DiGioia *et al*., 1996) es un sistema que utiliza técnicas de visión por computador para crear un modelo tridimensional de la anatomía interna del paciente y después utiliza un control robotizado para guiar el implante de prótesis de cadera.

Procesamiento de lenguaje y resolución de problemas: PROVER B (Littman *et al*., 1999) es un programa informático que resuelve crucigramas mejor que la mayoría de los humanos, utilizando restricciones en programas de relleno de palabras, una gran base de datos de crucigramas, y varias fuentes de información como diccionarios y bases de datos *online*, que incluyen la lista de películas y los actores que intervienen en ellas, entre otras cosas. Por ejemplo, determina que la pista «Historia de Niza» se puede resolver con «ETAGE» ya que su base de datos incluye el par pista/solución «Historia en Francia/ETAGE» y porque reconoce que los patrones «Niza X» y «X en Francia» a menudo tienen la misma solución. El programa no sabe que Niza es una ciudad de Francia, pero es capaz de resolver el puzle.

Estos son algunos de los ejemplos de sistemas de inteligencia artificial que existen hoy en día. No se trata de magia o ciencia ficción, son más bien ciencia, ingeniería y matemáticas, para los que este libro proporciona una introducción.

1.5 Resumen

En este capítulo se define la IA y se establecen los antecedentes culturales que han servido de base. Algunos de los aspectos más destacables son:

- Cada uno tiene una visión distinta de lo que es la IA. Es importante responder a las dos preguntas siguientes: ¿Está interesado en el razonamiento y el comportamiento? ¿Desea modelar seres humanos o trabajar a partir de un ideal estándar?
- En este libro se adopta el criterio de que la inteligencia tiene que ver principalmente con las **acciones racionales**. Desde un punto de vista ideal, un **agente inteligente** es aquel que emprende la mejor acción posible ante una situación dada. Se estudiará el problema de la construcción de agentes que sean inteligentes en este sentido.
- Los filósofos (desde el año 400 a.C.) facilitaron el poder imaginar la IA, al concebir la idea de que la mente es de alguna manera como una máquina que funciona a partir del conocimiento codificado en un lenguaje interno, y al considerar que el pensamiento servía para seleccionar la acción a llevar a cabo.
- Las matemáticas proporcionaron las herramientas para manipular tanto las aseveraciones de certeza lógicas, como las inciertas de tipo probabilista. Asimismo,

prepararon el terreno para un entendimiento de lo que es el cálculo y el razonamiento con algoritmos.

- Los economistas formalizaron el problema de la toma de decisiones para maximizar los resultados esperados.
- Los psicólogos adoptaron la idea de que los humanos y los animales podían considerarse máquinas de procesamiento de información. Los lingüistas demostraron que el uso del lenguaje se ajusta a ese modelo.
- Los informáticos proporcionaron los artefactos que hicieron posible la aplicación de la IA. Los programas de IA tienden a ser extensos y no podrían funcionar sin los grandes avances en velocidad y memoria aportados por la industria informática.
- La teoría de control se centra en el diseño de dispositivos que actúan de forma óptima con base en la retroalimentación que reciben del entorno en el que están inmersos. Inicialmente, las herramientas matemáticas de la teoría de control eran bastante diferentes a las técnicas que utilizaba la IA, pero ambos campos se están acercando.
- Las historia de la IA ha pasado por ciclos de éxito, injustificado optimismo y consecuente desaparición de entusiasmo y apoyos financieros. También ha habido ciclos caracterizados por la introducción de enfoques nuevos y creativos y de un perfeccionamiento sistemático de los mejores.
- La IA ha avanzado más rápidamente en la década pasada debido al mayor uso del método científico en la experimentación y comparación de propuestas.
- Los avances recientes logrados en el entendimiento de las bases teóricas de la inteligencia han ido aparejados con las mejoras realizadas en la optimización de los sistemas reales. Los subcampos de la IA se han integrado más y la IA ha encontrado elementos comunes con otras disciplinas.

Notas bibliográficas e históricas

El estatus metodológico de la inteligencia artificial se ha investigado en *The Sciences of the Artificial*, escrito por Herb Simon (1981), en el cual se analizan áreas de investigación interesadas en el desarrollo de artefactos complejos. Explica cómo la IA se puede ver como ciencia y matemática. Cohen (1995) proporciona una visión de la metodología experimental en el marco de la IA. Ford y Hayes (1995) presentan una revisión crítica de la utilidad de la Prueba de Turing.

Artificial Intelligence: The Very Idea, de John Haugeland (1985), muestra una versión amena de los problemas prácticos y filosóficos de la IA. La ciencia cognitiva está bien descrita en varios textos recientes (Johnson-Laird, 1988; Stillings *et al.*, 1995; Thagard, 1996) y en la *Encyclopedia of the Cognitive Sciences* (Wilson y Keil, 1999). Baker (1989) cubre la parte sintáctica de la lingüística moderna, y Chierchia y McConnell-Ginet (1990) la semántica. Jurafsky y Martin (2000) revisan la lingüística computacional.

Los primeros trabajos en el campo de la IA se citan en *Computers and Thought* (1963), de Feigenbaum y Feldman, en *Semantic Information Processing* de Minsky, y en la serie *Machine Intelligence*, editada por Donald Michie. Webber y Nilsson (1981)

y Luger (1995) han recogido una nutrida cantidad de artículos influyentes. Los primeros artículos sobre redes neuronales están reunidos en *Neurocomputing* (Anderson y Rosenfeld, 1988). La *Encyclopedia of AI* (Shapiro, 1992) contiene artículos de investigación sobre prácticamente todos los temas de IA. Estos artículos son muy útiles para iniciarse en las diversas áreas presentes en la literatura científica.

El trabajo más reciente se encuentra en las actas de las mayores conferencias de IA: la *International Joint Conference on AI* (IJCAI), de carácter bianual, la *European Conference on AI* (ECAI), de carácter bianual, y la *National Conference on AI*, conocida normalmente como AAAI por la organización que la patrocina. Las revistas científicas que presentan aspectos generales de la IA más importantes son *Artificial Intelligence, Computational Intelligence, IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Intelligent Systems*, y la revista electrónica *Journal of Artificial Intelligence Research*. Hay también numerosas revistas y conferencias especializadas en áreas concretas, que se mencionarán en los capítulos apropiados. La asociaciones profesionales de IA más importantes son la American Association for Artificial Intelligence (AAAI), la ACM Special Interest Group in Artificial Intelligence (SIGART), y la Society for Artificial Intelligence and Simulation of Behaviour (AISB). La revista *AI Magazine* de AAAI contiene muchos artículos de interés general y manuales, y su página Web, aaai.org contiene noticias e información de referencia.

EJERCICIOS

El propósito de los siguientes ejercicios es estimular la discusión, y algunos de ellos se podrían utilizar como proyectos. Alternativamente, se podría hacer un esfuerzo inicial para solucionarlos ahora, de forma que una vez se haya leído todo el libro se puedan revisar estos primeros intentos.

1.1 Defina con sus propias palabras: (a) inteligencia, (b) inteligencia artificial, (c) agente.

1.2 Lea el artículo original de Turing sobre IA (Turing, 1950). En él se comentan algunas objeciones potenciales a su propuesta y a su prueba de inteligencia. ¿Cuáles de estas objeciones tiene todavía validez? ¿Son válidas sus refutaciones? ¿Se le ocurren nuevas objeciones a esta propuesta teniendo en cuenta los desarrollos realizados desde que se escribió el artículo? En el artículo, Turing predijo que para el año 2000 sería probable que un computador tuviera un 30 por ciento de posibilidades de superar una Prueba de Turing dirigida por un evaluador inexperto con una duración de cinco minutos. ¿Considera razonable lo anterior en el mundo actual? ¿Y en los próximos 50 años?

1.3 Todos los años se otorga el premio Loebner al programa que lo hace mejor en una Prueba de Turing concreta. Investigue y haga un informe sobre el último ganador del premio Loebner. ¿Qué técnica utiliza? ¿Cómo ha hecho que progrese la investigación en el campo de la IA?

1.4 Hay clases de problemas bien conocidos que son intratables para los computadores, y otras clases sobre los cuales un computador no pueda tomar una decisión. ¿Quiere esto decir que es imposible lograr la IA?

1.5 Supóngase que se extiende ANALOGY, el programa de Evans, como para alcanzar una puntuación de 200 en una prueba normal de cociente de inteligencia. ¿Quiere decir lo anterior que se ha creado un programa más inteligente que un ser humano? Explíquese.

1.6 ¿Cómo puede la introspección (revisión de los pensamientos íntimos) ser inexacta? ¿Se puede estar equivocado sobre lo que se cree? Discútase.

1.7 Consulte en la literatura existente sobre la IA si alguna de las siguientes tareas se puede efectuar con computadores:

a) Jugar una partida de tenis de mesa (ping-pong) decentemente.
b) Conducir un coche en el centro del Cairo.
c) Comprar comestibles para una semana en el mercado.
d) Comprar comestibles para una semana en la web.
e) Jugar una partida de *bridge* decentemente a nivel de competición.
f) Descubrir y demostrar nuevos teoremas matemáticos.
g) Escribir intencionadamente una historia divertida.
h) Ofrecer asesoría legal competente en un área determinada.
i) Traducir inglés hablado al sueco hablado en tiempo real.
j) Realizar una operación de cirugía compleja.

En el caso de las tareas que no sean factibles de realizar en la actualidad, trate de describir cuáles son las dificultades y calcule para cuándo se podrán superar.

1.8 Algunos autores afirman que la percepción y las habilidades motoras son la parte más importante de la inteligencia y que las capacidades de «alto nivel» son más bien parásitas (simples añadidos a las capacidades básicas). Es un hecho que la mayor parte de la evolución y que la mayor parte del cerebro se han concentrado en la percepción y las habilidades motoras, en tanto la IA ha descubierto que tareas como juegos e inferencia lógica resultan más sencillas, en muchos sentidos, que percibir y actuar en el mundo real. ¿Consideraría usted que ha sido un error la concentración tradicional de la IA en las capacidades cognitivas de alto nivel?

1.9 ¿Por qué la evolución tiende a generar sistemas que actúan racionalmente? ¿Qué objetivos deben intentar alcanzar estos sistemas?

1.10 ¿Son racionales las acciones reflejas (como retirar la mano de una estufa caliente)? ¿Son inteligentes?

1.11 «En realidad los computadores no son inteligentes, hacen solamente lo que le dicen los programadores». ¿Es cierta la última aseveración, e implica a la primera?

1.12 «En realidad los animales no son inteligentes, hacen solamente lo que le dicen sus genes». ¿Es cierta la última aseveración, e implica a la primera?

1.13 «En realidad los animales, los humanos y los computadores no pueden ser inteligentes, ellos sólo hacen lo que los átomos que los forman les dictan siguiendo las leyes de la física». ¿Es cierta la última aseveración, e implica a la primera?

2 Agentes inteligentes

Donde se discutirá la naturaleza de los agentes ideales, sus diversos hábitats y las formas de organizar los tipos de agentes existentes.

El Capítulo 1 identifica el concepto de **agente racional** como central en la perspectiva de la inteligencia artificial que presenta este libro. Esta noción se concreta más a lo largo de este capítulo. Se mostrará como el concepto de racionalidad se puede aplicar a una amplia variedad de agentes que operan en cualquier medio imaginable. En el libro, la idea es utilizar este concepto para desarrollar un pequeño conjunto de principios de diseño que sirvan para construir agentes útiles, sistemas que se puedan llamar razonablemente **inteligentes**.

Se comienza examinando los agentes, los medios en los que se desenvuelven, y la interacción entre éstos. La observación de que algunos agentes se comportan mejor que otros nos lleva naturalmente a la idea de agente racional, aquel que se comporta tan bien como puede. La forma de actuar del agente depende de la naturaleza del medio; algunos hábitats son más complejos que otros. Se proporciona una categorización cruda del medio y se muestra cómo las propiedades de un hábitat influyen en el diseño de agentes adecuados para ese entorno. Se presenta un número de «esquemas» básicos para el diseño de agentes, a los que se dará cuerpo a lo largo del libro.

2.1 Agentes y su entorno

MEDIOAMBIENTE

Un **agente** es cualquier cosa capaz de percibir su **medioambiente** con la ayuda de **sensores** y actuar en ese medio utilizando **actuadores**[1]. La Figura 2.1 ilustra esta idea sim-

[1] Se usa este término para indicar el elemento que reacciona a un estímulo realizando una acción (*N. del RT*).

Figura 2.1 Los agentes interactúan con el medioambiente mediante sensores y efectores.

SENSOR

ACTUADOR

ple. Un agente humano tiene ojos, oídos y otros órganos sensoriales además de manos, piernas, boca y otras partes del cuerpo para actuar. Un agente robot recibe pulsaciones del teclado, archivos de información y paquetes vía red a modo de entradas sensoriales y actúa sobre el medio con mensajes en el monitor, escribiendo ficheros y enviando paquetes por la red. Se trabajará con la hipótesis general de que cada agente puede percibir sus propias acciones (pero no siempre sus efectos).

PERCEPCIÓN

SECUENCIA DE PERCEPTORES

El término **percepción** se utiliza en este contexto para indicar que el agente puede recibir entradas en cualquier instante. La **secuencia de percepciones** de un agente refleja el historial completo de lo que el agente ha recibido. En general, *un agente tomará una decisión en un momento dado dependiendo de la secuencia completa de percepciones hasta ese instante*. Si se puede especificar qué decisión tomará un agente para cada una de las posibles secuencias de percepciones, entonces se habrá explicado más o menos todo lo que se puede decir de un agente. En términos matemáticos se puede decir que el comportamiento del agente viene dado por la **función del agente** que proyecta una percepción dada en una acción.

FUNCIÓN DEL AGENTE

La función que describe el comportamiento de un agente se puede presentar en *forma de tabla*; en la mayoría de los casos esta tabla sería muy grande (infinita a menos que se limite el tamaño de la secuencia de percepciones que se quiera considerar). Dado un agente, con el que se quiera experimentar, se puede, en principio, construir esta tabla teniendo en cuenta todas las secuencias de percepción y determinando qué acción lleva a cabo el agente en respuesta[2]. La tabla es, por supuesto, una caracterización *externa* del agente. *Inicialmente*, la función del agente para un agente artificial se imple-

[2] Si el agente selecciona la acción de manera aleatoria, entonces sería necesario probar cada secuencia muchas veces para identificar la probabilidad de cada acción. Se puede pensar que actuar de manera aleatoria es ridículo, pero como se verá posteriormente puede ser muy inteligente.

PROGRAMA DEL AGENTE

mentará mediante el **programa del agente**. Es importante diferenciar estas dos ideas. La función del agente es una descripción matemática abstracta; el programa del agente es una implementación completa, que se ejecuta sobre la arquitectura del agente.

Para ilustrar esta idea se utilizará un ejemplo muy simple, el mundo de la aspiradora presentado en la Figura 2.2. Este mundo es tan simple que se puede describir todo lo que en él sucede; es un mundo hecho a medida, para el que se pueden inventar otras variaciones. Este mundo en particular tiene solamente dos localizaciones: cuadrícula *A* y *B*. La aspiradora puede percibir en qué cuadrante se encuentra y si hay suciedad en él. Puede elegir si se mueve hacia la izquierda, derecha, aspirar la suciedad o no hacer nada. Una función muy simple para el agente vendría dada por: si la cuadrícula en la que se encuentra está sucia, entonces aspirar, de otra forma cambiar de cuadrícula. Una muestra parcial de la función del agente representada en forma de tabla aparece en la Figura 2.3. Un programa de agente simple para esta función de agente se mostrará posteriormente en la Figura 2.8.

Figura 2.2 El mundo de la aspiradora con dos localizaciones solamente.

Secuencia de percepciones	Acción
[*A, Limpio*]	*Derecha*
[*A, Sucio*]	*Aspirar*
[*B, Limpio*]	*Izquierda*
[*B, Sucio*]	*Aspirar*
[*A, Limpio*], [*A, Limpio*]	*Derecha*
[*A, Limpio*], [*A, Sucio*]	*Aspirar*
—	—
—	—
—	—
[*A, Limpio*], [*A, Limpio*], [*A, Limpio*]	*Derecha*
[*A, Limpio*], [*A, Limpio*], [*A, Sucio*]	*Aspirar*
—	—
—	—
—	—

Figura 2.3 Tabla parcial de una función de agente sencilla para el mundo de la aspiradora que se muestra en la Figura 2.2.

Revisando la Figura 2.3, se aprecia que se pueden definir varios agentes para el mundo de la aspiradora simplemente rellenando la columna de la derecha de formas distintas. La pregunta obvia, entonces es: *¿cuál es la mejor forma de rellenar una tabla*? En otras palabras, ¿qué hace que un agente sea bueno o malo, inteligente o estúpido? Estas preguntas se responden en la siguiente sección.

Antes de terminar esta sección, es necesario remarcar que la noción de agente es supuestamente una herramienta para el análisis de sistemas, y no una caracterización absoluta que divida el mundo entre agentes y no agentes. Se puede ver una calculadora de mano como un agente que elige la acción de mostrar «4» en la pantalla, dada la secuencia de percepciones «2 + 2 =». Pero este análisis difícilmente puede mejorar nuestro conocimiento acerca de las calculadoras.

2.2 Buen comportamiento: el concepto de racionalidad

AGENTE RACIONAL

Un **agente racional** es aquel que hace lo correcto; en términos conceptuales, cada elemento de la tabla que define la función del agente se tendría que rellenar correctamente. Obviamente, hacer lo correcto es mejor que hacer algo incorrecto, pero ¿qué significa hacer lo correcto? Como primera aproximación, se puede decir que lo correcto es aquello que permite al agente obtener un resultado mejor. Por tanto, se necesita determinar una forma de medir el éxito. Ello, junto a la descripción del entorno y de los sensores y actuadores del agente, proporcionará una especificación completa de la tarea que desempeña el agente. Dicho esto, ahora es posible definir de forma más precisa qué significa la racionalidad.

Medidas de rendimiento

MEDIDAS DE RENDIMIENTO

Las **medidas de rendimiento** incluyen los criterios que determinan el éxito en el comportamiento del agente. Cuando se sitúa un agente en un medio, éste genera una secuencia de acciones de acuerdo con las percepciones que recibe. Esta secuencia de acciones hace que su hábitat pase por una secuencia de estados. Si la secuencia es la deseada, entonces el agente habrá actuado correctamente. Obviamente, no hay una única medida adecuada para todos los agentes. Se puede preguntar al agente por su opinión subjetiva acerca de su propia actuación, pero muchos agentes serían incapaces de contestar, y otros podrían engañarse a sí mismos[3]. Por tanto hay que insistir en la importancia de utilizar medidas de rendimiento objetivas, que normalmente determinará el diseñador encargado de la construcción del agente.

Si retomamos el ejemplo de la aspiradora de la sección anterior, se puede proponer utilizar como medida de rendimiento la cantidad de suciedad limpiada en un período de

[3] Los agentes humanos son conocidos en particular por su «acidez», hacen creer que no quieren algo después de no haberlo podido conseguir, por ejemplo, «Ah bueno, de todas formas no quería ese estúpido Premio Nobel».

ocho horas. Con agentes racionales, por supuesto, se obtiene lo que se demanda. Un agente racional puede maximizar su medida de rendimiento limpiando la suciedad, tirando la basura al suelo, limpiándola de nuevo, y así sucesivamente. Una medida de rendimiento más adecuada recompensaría al agente por tener el suelo limpio. Por ejemplo, podría ganar un punto por cada cuadrícula limpia en cada período de tiempo (quizás habría que incluir algún tipo de penalización por la electricidad gastada y el ruido generado). *Como regla general, es mejor diseñar medidas de utilidad de acuerdo con lo que se quiere para el entorno, más que de acuerdo con cómo se cree que el agente debe comportarse*.

La selección de la medida de rendimiento no es siempre fácil. Por ejemplo, la noción de «suelo limpio» del párrafo anterior está basada en un nivel de limpieza medio a lo largo del tiempo. Además, este nivel medio de limpieza se puede alcanzar de dos formas diferentes, llevando a cabo una limpieza mediocre pero continua o limpiando en profundidad, pero realizando largos descansos. La forma más adecuada de hacerlo puede venir dada por la opinión de un encargado de la limpieza profesional, pero en realidad es una cuestión filosófica profunda con fuertes implicaciones. ¿Qué es mejor, una vida temeraria con altos y bajos, o una existencia segura pero aburrida? ¿Qué es mejor, una economía en la que todo el mundo vive en un estado de moderada pobreza o una en la que algunos viven en la abundancia y otros son muy pobres? Estas cuestiones se dejan como ejercicio para los lectores diligentes.

Racionalidad

La racionalidad en un momento determinado depende de cuatro factores:

- La medida de rendimiento que define el criterio de éxito.
- El conocimiento del medio en el que habita acumulado por el agente.
- Las acciones que el agente puede llevar a cabo.
- La secuencia de percepciones del agente hasta este momento.

DEFINICIÓN DE AGENTE RACIONAL

Esto nos lleva a la **definición de agente racional**:

> *En cada posible secuencia de percepciones, un agente racional deberá emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado.*

Considerando que el agente aspiradora limpia una cuadrícula si está sucia y se mueve a la otra si no lo está (ésta es la función del agente que aparece en la tabla de la Figura 2.3), ¿se puede considerar racional? ¡Depende! Primero, se debe determinar cuál es la medida de rendimiento, qué se conoce del entorno, y qué sensores y actuadores tiene el agente. Si asumimos que:

- La medida de rendimiento premia con un punto al agente por cada recuadro limpio en un período de tiempo concreto, a lo largo de una «vida» de 1.000 períodos.
- La «geografía» del medio se conoce *a priori* (Figura 2.2), pero que la distribución de la suciedad y la localización inicial del agente no se conocen. Las cuadrículas se mantienen limpias y aspirando se limpia la cuadrícula en que se encuentre el agente. Las acciones *Izquierda* y *Derecha* mueven al agente hacia la izquierda y

derecha excepto en el caso de que ello pueda llevar al agente fuera del recinto, en este caso el agente permanece donde se encuentra.
- Las únicas acciones permitidas son *Izquierda*, *Derecha*, *Aspirar* y *NoOp* (no hacer nada).
- El agente percibe correctamente su localización y si esta localización contiene suciedad.

Puede afirmarse que *bajo estas circunstancias* el agente es verdaderamente racional; el rendimiento que se espera de este agente es por lo menos tan alto como el de cualquier otro agente. El Ejercicio 2.4 pide que se pruebe este hecho.

Fácilmente se puede observar que el agente puede resultar irracional en circunstancias diferentes. Por ejemplo, cuando toda la suciedad se haya eliminado el agente oscilará innecesariamente hacia delante y atrás; si la medida de rendimiento incluye una penalización de un punto por cada movimiento hacia la derecha e izquierda, la respuesta del agente será pobre. Un agente más eficiente no hará nada si está seguro de que todas las cuadrículas están limpias. Si una cuadrícula se ensucia de nuevo, el agente debe identificarlo en una de sus revisiones ocasionales y limpiarla. Si no se conoce la geografía del entorno, el agente tendrá que explorarla y no quedarse parado en las cuadrículas *A* y *B*. El Ejercicio 2.4 pide que se diseñen agentes para estos casos.

Omnisciencia, aprendizaje y autonomía

OMNISCIENCIA

Es necesario tener cuidado al distinguir entre racionalidad y **omnisciencia**. Un agente omnisciente conoce el resultado de su acción y actúa de acuerdo con él; sin embargo, en realidad la omnisciencia no es posible. Considerando el siguiente ejemplo: estoy paseando por los Campos Elíseos y veo un amigo al otro lado de la calle. No hay tráfico alrededor y no tengo ningún compromiso, entonces, actuando racionalmente, comenzaría a cruzar la calle. Al mismo tiempo, a 33.000 pies de altura, se desprende la puerta de un avión[4], y antes de que termine de cruzar al otro lado de la calle me encuentro aplastado. ¿Fue irracional cruzar la calle? Sería de extrañar que en mi nota necrológica apareciera «Un idiota intentando cruzar la calle».

Este ejemplo muestra que la racionalidad no es lo mismo que la perfección. La racionalidad maximiza el rendimiento esperado, mientras la perfección maximiza el resultado real. Alejarse de la necesidad de la perfección no es sólo cuestión de hacer justicia con los agentes. El asunto es que resulta imposible diseñar un agente que siempre lleve a cabo, de forma sucesiva, las mejores acciones después de un acontecimiento, a menos que se haya mejorado el rendimiento de las bolas de cristal o las máquinas de tiempo.

La definición propuesta de racionalidad no requiere omnisciencia, ya que la elección racional depende sólo de la secuencia de percepción hasta *la fecha*. Es necesario asegurase de no haber permitido, por descuido, que el agente se dedique decididamente a llevar a cabo acciones poco inteligentes. Por ejemplo, si el agente no mirase a ambos lados de la calle antes de cruzar una calle muy concurrida, entonces su secuencia de per-

[4] *Véase* N. Henderson, «New door latches urged for Boeing 747 jumbo jets» (es urgente dotar de nuevas cerraduras a las puertas de los Boeing jumbo 747), *Washington Post*, 24 de agosto de 1989.

cepción no le indicaría que se está acercando un gran camión a gran velocidad. ¿La definición de racionalidad nos está indicando que está bien cruzar la calle? ¡Todo lo contrario! Primero, no sería racional cruzar la calle sólo teniendo esta secuencia de percepciones incompleta: el riesgo de accidente al cruzarla sin mirar es demasiado grande. Segundo, un agente racional debe elegir la acción de «mirar» antes de intentar cruzar la calle, ya que el mirar maximiza el rendimiento esperado. Llevar a cabo acciones *con la intención de modificar percepciones futuras*, en ocasiones proceso denominado **recopilación de información**, es una parte importante de la racionalidad y se comenta en profundidad en el Capítulo 16. Un segundo ejemplo de recopilación de información lo proporciona la **exploración** que debe llevar a cabo el agente aspiradora en un medio inicialmente desconocido.

RECOPILACIÓN DE INFORMACIÓN

EXPLORACIÓN

La definición propuesta implica que el agente racional no sólo recopile información, sino que **aprenda** lo máximo posible de lo que está percibiendo. La configuración inicial del agente puede reflejar un conocimiento preliminar del entorno, pero a medida que el agente adquiere experiencia éste puede modificarse y aumentar. Hay casos excepcionales en los que se conoce totalmente el entorno *a priori*. En estos casos, el agente no necesita percibir y aprender; simplemente actúa de forma correcta. Por supuesto, estos agentes son muy frágiles. Considérese el caso del humilde escarabajo estercolero. Después de cavar su nido y depositar en él su huevos, tomó una bola de estiércol de una pila cercana para tapar su entrada. Si *durante el trayecto* se le quita la bola, el escarabajo continuará su recorrido y hará como si estuviera tapando la entrada del nido, sin tener la bola y sin darse cuanta de ello. La evolución incorporó una suposición en la conducta del escarabajo, y cuando se viola, el resultado es un comportamiento insatisfactorio. La avispa cavadora es un poco más inteligente. La avispa hembra cavará una madriguera, saldrá de ella, picará a una oruga y la llevará a su madriguera, se introducirá en la madriguera para comprobar que todo está bien, arrastrará la oruga hasta el fondo y pondrá sus huevos. La oruga servirá como fuente de alimento cuando los huevos se abran. Hasta ahora todo bien, pero si un entomólogo desplaza la oruga unos centímetros fuera cuando la avispa está revisando la situación, ésta volverá a la etapa de «arrastre» que figura en su plan, y continuará con el resto del plan sin modificación alguna, incluso después de que se intervenga para desplazar la oruga. La avispa cavadora no es capaz de aprender que su plan innato está fallando, y por tanto no lo cambiará.

APRENDIZAJE

Los agentes con éxito dividen las tareas de calcular la función del agente en tres períodos diferentes: cuando se está diseñando el agente, y están los diseñadores encargados de realizar algunos de estos cálculos; cuando está pensando en la siguiente operación, el agente realiza más cálculos; y cuando está aprendiendo de la experiencia, el agente lleva a cabo más cálculos para decidir cómo modificar su forma de comportarse.

AUTONOMÍA

Se dice que un agente carece de **autonomía** cuando se apoya más en el conocimiento inicial que le proporciona su diseñador que en sus propias percepciones. Un agente racional debe ser autónomo, debe saber aprender a determinar cómo tiene que compensar el conocimiento incompleto o parcial inicial. Por ejemplo, el agente aspiradora que aprenda a prever dónde y cuándo aparecerá suciedad adicional lo hará mejor que otro que no aprenda. En la práctica, pocas veces se necesita autonomía completa desde el comienzo: cuando el agente haya tenido poca o ninguna experiencia, tendrá que actuar de forma aleatoria a menos que el diseñador le haya proporcionado ayuda. Así, de la

misma forma que la evolución proporciona a los animales sólo los reactivos necesarios para que puedan sobrevivir lo suficiente para aprender por ellos mismos, sería razonable proporcionar a los agentes que disponen de inteligencia artificial un conocimiento inicial, así como de la capacidad de aprendizaje. Después de las suficientes experiencias interaccionando con el entorno, el comportamiento del agente racional será efectivamente *independiente* del conocimiento que poseía inicialmente. De ahí, que la incorporación del aprendizaje facilite el diseño de agentes racionales individuales que tendrán éxito en una gran cantidad de medios.

2.3 La naturaleza del entorno

ENTORNOS DE TRABAJO

Ahora que se tiene una definición de racionalidad, se está casi preparado para pensar en la construcción de agentes racionales. Primero, sin embargo, hay que centrarse en los **entornos de trabajo**, que son esencialmente los «problemas» para los que los agentes racionales son las «soluciones». Para ello se comienza mostrando cómo especificar un entorno de trabajo, ilustrando el proceso con varios ejemplos. Posteriormente se mostrará que el entorno de trabajo ofrece diferentes posibilidades, de forma que cada una de las posibilidades influyen directamente en el diseño del programa del agente.

Especificación del entorno de trabajo

REAS

En la discusión de la racionalidad de un agente aspiradora simple, hubo que especificar las medidas de rendimiento, el entorno, y los actuadores y sensores del agente. Todo ello forma lo que se llama el **entorno de trabajo**, para cuya denominación se utiliza el acrónimo **REAS** (**R**endimiento, **E**ntorno, **A**ctuadores, **S**ensores). En el diseño de un agente, el primer paso debe ser siempre especificar el entorno de trabajo de la forma más completa posible.

El mundo de la aspiradora fue un ejemplo simple; considérese ahora un problema más complejo: un taxista automático. Este ejemplo se utilizará a lo largo del capítulo. Antes de alarmar al lector, conviene aclarar que en la actualidad la construcción de un taxi automatizado está fuera del alcance de la tecnología actual. *Véase* en la página 31 la descripción de un robot conductor que ya existe en la actualidad, o lea las actas de la conferencia *Intelligent Transportation Systems*. La tarea de conducir un automóvil, en su totalidad, es extremadamente *ilimitada*. No hay límite en cuanto al número de nuevas combinaciones de circunstancias que pueden surgir (por esta razón se eligió esta actividad en la presente discusión). La Figura 2.4 resume la descripción REAS para el entorno de trabajo del taxi. El próximo párrafo explica cada uno de sus elementos en más detalle.

Primero, ¿cuál es el **entorno de trabajo** en el que el taxista automático aspira a conducir? Dentro de las cualidades deseables que debería tener se incluyen el que llegue al destino correcto; que minimice el consumo de combustible; que minimice el tiempo de viaje y/o coste; que minimice el número de infracciones de tráfico y de molestias a otros conductores; que maximice la seguridad, la comodidad del pasajero y el

Tipo de agente	Medidas de rendimiento	Entorno	Actuadores	Sensores
Taxista	Seguro, rápido, legal, viaje confortable, maximización del beneficio	Carreteras, otro tráfico, peatones, clientes	Dirección, acelerador, freno, señal, bocina, visualizador	Cámaras, sónar, velocímetro, GPS, tacómetro, visualizador de la aceleración, sensores del motor, teclado

Figura 2.4 Descripción REAS del entorno de trabajo de un taxista automático.

beneficio. Obviamente, alguno de estos objetivos entran en conflicto por lo que habrá que llegar a acuerdos.

Siguiente, ¿cuál es el **entorno** en el que se encontrará el taxi? Cualquier taxista debe estar preparado para circular por distintas carreteras, desde caminos rurales y calles urbanas hasta autopistas de 12 carriles. En las carreteras se pueden encontrar con tráfico, peatones, animales, obras, coches de policía, charcos y baches. El taxista también tiene que comunicarse tanto con pasajeros reales como potenciales. Hay también elecciones opcionales. El taxi puede operar en California del Sur, donde la nieve es raramente un problema, o en Alaska, donde raramente no lo es. Puede conducir siempre por la derecha, o puede ser lo suficientemente flexible como para que circule por la izquierda cuando se encuentre en el Reino Unido o en Japón. Obviamente, cuanto más restringido esté el entorno, más fácil será el problema del diseño.

Los **actuadores** disponibles en un taxi automático serán más o menos los mismos que los que tiene a su alcance un conductor humano: el control del motor a través del acelerador y control sobre la dirección y los frenos. Además, necesitará tener una pantalla de visualización o un sintetizador de voz para responder a los pasajeros, y quizás algún mecanismo para comunicarse, educadamente o de otra forma, con otros vehículos.

Para alcanzar sus objetivos en el entorno en el que circula, el taxi necesita saber dónde está, qué otros elementos están en la carretera, y a qué velocidad circula. Sus **sensores** básicos deben, por tanto, incluir una o más cámaras de televisión dirigidas, un velocímetro y un tacómetro. Para controlar el vehículo adecuadamente, especialmente en las curvas, debe tener un acelerador; debe conocer el estado mecánico del vehículo, de forma que necesitará sensores que controlen el motor y el sistema eléctrico. Debe tener instrumentos que no están disponibles para un conductor medio: un sistema de posicionamiento global vía satélite (GPS) para proporcionarle información exacta sobre su posición con respecto a un mapa electrónico, y sensores infrarrojos o sonares para detectar las distancias con respecto a otros coches y obstáculos. Finalmente, necesitará un teclado o micrófono para que el pasajero le indique su destino.

La Figura 2.5 muestra un esquema con los elementos REAS básicos para diferentes clases de agentes adicionales. Más ejemplos aparecerán en el Ejercicio 2.5. Puede sorprender a algunos lectores que se incluya en la lista de tipos de agente algunos programas que operan en la totalidad del entorno artificial definido por las entradas del teclado y los caracteres impresos en el monitor. «Seguramente», nos podamos pregun-

Tipo de agente	Medidas de rendimiento	Entorno	Actuadores	Sensores
Sistema de diagnóstico médico	Pacientes sanos, reducir costes, demandas	Pacientes, hospital, personal	Visualizar preguntas, pruebas, diagnósticos, tratamientos, casos	Teclado para la entrada de síntomas, conclusiones, respuestas de pacientes
Sistema de análisis de imágenes de satélites	Categorización de imagen correcta	Conexión con el satélite en órbita	Visualizar la categorización de una escena	Matriz de pixels de colores
Robot para la selección de componentes	Porcentaje de componentes clasificados en los cubos correctos	Cinta transportadora con componentes, cubos	Brazo y mano articulados	Cámara, sensor angular
Controlador de una refinería	Maximizar la pureza, producción y seguridad	Refinería, operadores	Válvulas, bombas, calentadores, monitores	Temperatura, presión, sensores químicos
Tutor de inglés interactivo	Maximizar la puntuación de los estudiantes en los exámenes	Conjunto de estudiantes, agencia examinadora	Visualizar los ejercicios, sugerencias, correcciones	Teclado de entrada

Figura 2.5 Ejemplos de tipos de agentes y sus descripciones REAS.

tar, «¿este no es un entorno real, verdad?». De hecho, lo que importa no es la distinción entre un medio «real» y «artificial», sino la complejidad de la relación entre el comportamiento del agente, la secuencia de percepción generada por el medio y la medida de rendimiento. Algunos entornos «reales» son de hecho bastante simples. Por ejemplo, un robot diseñado para inspeccionar componentes según pasan por una cinta transportadora puede hacer uso de varias suposiciones simples: que la cinta siempre estará iluminada, que conocerá todos los componentes que circulen por la cinta, y que hay solamente dos acciones (aceptar y rechazar).

AGENTES *SOFTWARE*

SOFTBOTS

En contraste, existen algunos **agentes *software*** (o robots *software* o **softbots**) en entornos ricos y prácticamente ilimitados. Imagine un softbot diseñado para pilotar el simulador de vuelo de un gran avión comercial. El simulador constituye un medio muy detallado y complejo que incluye a otros aviones y operaciones de tierra, y el agente *software* debe elegir, en tiempo real, una de entre un amplio abanico de posibilidades. O imagine un robot diseñado para que revise fuentes de información en Internet y para que muestre aquellas que sean interesantes a sus clientes. Para lograrlo, deberá poseer cierta habilidad en el procesamiento de lenguaje natural, tendrá que aprender qué es lo que le interesa a cada cliente, y tendrá que ser capaz de cambiar sus planes dinámica-

mente, por ejemplo, cuando se interrumpa la conexión con una fuente de información o cuando aparezca una nueva. Internet es un medio cuya complejidad rivaliza con la del mundo físico y entre cuyos habitantes se pueden incluir muchos agentes artificiales.

Propiedades de los entornos de trabajo

El rango de los entornos de trabajo en los que se utilizan técnicas de IA es obviamente muy grande. Sin embargo, se puede identificar un pequeño número de dimensiones en las que categorizar estos entornos. Estas dimensiones determinan, hasta cierto punto, el diseño más adecuado para el agente y la utilización de cada una de las familias principales de técnicas en la implementación del agente. Primero se enumeran la dimensiones, y después se analizan varios entornos de trabajo para ilustrar estas ideas. Las definiciones dadas son informales; capítulos posteriores proporcionan definiciones más precisas y ejemplos de cada tipo de entorno.

TOTALMENTE OBSEVABLE

- **Totalmente observable** *vs.* **parcialmente observable.**
  Si los sensores del agente le proporcionan acceso al estado completo del medio en cada momento, entonces se dice que el entorno de trabajo es totalmente observable[5]. Un entorno de trabajo es, efectivamente, totalmente observable si los sensores detectan todos los aspectos que son relevantes en la toma de decisiones; la relevancia, en cada momento, depende de las medidas de rendimiento. Entornos totalmente observables son convenientes ya que el agente no necesita mantener ningún estado interno para saber qué sucede en el mundo. Un entorno puede ser parcialmente observable debido al ruido y a la existencia de sensores poco exactos o porque los sensores no reciben información de parte del sistema, por ejemplo, un agente aspiradora con sólo un sensor de suciedad local no puede saber si hay suciedad en la otra cuadrícula, y un taxi automatizado no pude saber qué están pensando otros conductores.

DETERMINISTA

ESTOCÁSTICO

- **Determinista** *vs.* **estocástico.**
  Si el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente, entonces se dice que el entorno es determinista; de otra forma es estocástico. En principio, un agente no se tiene que preocupar de la incertidumbre en un medio totalmente observable y determinista. Sin embargo, si el medio es parcialmente observable entonces puede *parecer* estocástico. Esto es particularmente cierto si se trata de un medio complejo, haciendo difícil el mantener constancia de todos las aspectos observados. Así, a menudo es mejor pensar en entornos deterministas o estocásticos *desde el punto de vista del agente*. El agente taxi es claramente estocástico en este sentido, ya que no se puede predecir el comportamiento del tráfico exactamente; más aún, una rueda se puede reventar y un motor se puede gripar sin previo aviso. El mundo de la aspiradora es deter-

[5] La primera edición de este libro utiliza los términos **accesible** e **inaccesible** en vez de **total** y **parcialmente observable; no determinista** en vez de **estocástico;** y **no episódico** en vez de **secuencial.** La nueva terminología es más consistente con el uso establecido.

minista, como ya se describió, pero las variaciones pueden incluir elementos estocásticos como la aparición de suciedad aleatoria y un mecanismo de succión ineficiente (Ejercicio 2.12). Si el medio es determinista, excepto para las acciones de otros agentes, decimos que el medio es **estratégico**.

ESTRATÉGICO

EPISÓDICO

SECUENCIAL

- **Episódico** *vs.* **secuencial**[6]**.**
  En un entorno de trabajo episódico, la experiencia del agente se divide en episodios atómicos. Cada episodio consiste en la percepción del agente y la realización de una única acción posterior. Es muy importante tener en cuenta que el siguiente episodio no depende de las acciones que se realizaron en episodios previos. En los medios episódicos la elección de la acción en cada episodio depende sólo del episodio en sí mismo. Muchas tareas de clasificación son episódicas. Por ejemplo, un agente que tenga que seleccionar partes defectuosas en una cadena de montaje basa sus decisiones en la parte que está evaluando en cada momento, sin tener en cuenta decisiones previas; más aún, a la decisión presente no le afecta el que la próxima fase sea defectuosa. En entornos secuenciales, por otro lado, la decisión presente puede afectar a decisiones futuras. El ajedrez y el taxista son secuenciales: en ambos casos, las acciones que se realizan a corto plazo pueden tener consecuencias a largo plazo. Los medios episódicos son más simples que los secuenciales porque la gente no necesita pensar con tiempo.

ESTÁTICO

DINAMICO

- **Estático** *vs.* **dinámico.**
  Si el entorno puede cambiar cuando el agente está deliberando, entonces se dice que el entorno es dinámico para el agente; de otra forma se dice que es estático. Los medios estáticos son fáciles de tratar ya que el agente no necesita estar pendiente del mundo mientras está tomando una decisión sobre una acción, ni necesita preocuparse sobre el paso del tiempo. Los medios dinámicos, por el contrario, están preguntando continuamente al agente qué quiere hacer; si no se ha decidido aún, entonces se entiende que ha tomado la decisión de no hacer nada. Si el entorno no cambia con el paso del tiempo, pero el rendimiento del agente cambia, entonces se dice que el medio es **semidinámico**. El taxista es claramente dinámico: tanto los otros coches como el taxi se están moviendo mientras el algoritmo que guía la conducción indica qué es lo próximo a hacer. El ajedrez, cuando se juega con un reloj, es semideterminista. Los crucigramas son estáticos.

SEMIDINÁMICO

DISCRETO

CONTINUO

- **Discreto** *vs.* **continuo.**
  La distinción entre discreto y continuo se puede aplicar al *estado* del medio, a la forma en la que se maneja el *tiempo* y a las *percepciones* y *acciones* del agente. Por ejemplo, un medio con estados discretos como el del juego del ajedrez tiene un número finito de estados distintos. El ajedrez tiene un conjunto discreto de percepciones y acciones. El taxista conduciendo define un estado continuo y un problema de tiempo continuo: la velocidad y la ubicación del taxi y de los otros vehículos pasan por un rango de valores continuos de forma suave a lo largo del

[6] La palabra «secuencial» se utiliza también en el campo de la informática como antónimo de «paralelo». Los dos significados no están relacionados.

tiempo. Las conducción del taxista es también continua (ángulo de dirección, etc.). Las imágenes captadas por cámaras digitales son discretas, en sentido estricto, pero se tratan típicamente como representaciones continuas de localizaciones e intensidades variables.

AGENTE INDIVIDUAL

MULTIAGENTE

- **Agente individual** *vs.* **multiagente.**
  La distinción entre el entorno de un agente individual y el de un sistema multiagente puede parecer suficientemente simple. Por ejemplo, un agente resolviendo un crucigrama por sí mismo está claramente en un entorno de agente individual, mientras que un agente que juega al ajedrez está en un entorno con dos agentes. Sin embargo hay algunas diferencias sutiles. Primero, se ha descrito que una entidad *puede* percibirse como un agente, pero no se ha explicado qué entidades se *deben* considerar agentes. ¿Tiene el agente *A* (por ejemplo el agente taxista) que tratar un objeto *B* (otro vehículo) como un agente, o puede tratarse méramente como un objeto con un comportamiento estocástico, como las olas de la playa o las hojas que mueve el viento? La distinción clave está en identificar si el comportamiento de *B* está mejor descrito por la maximización de una medida de rendimiento cuyo valor depende del comportamiento de *A*. Por ejemplo, en el ajedrez, la entidad oponente *B* intenta maximizar su medida de rendimiento, la cual, según las reglas, minimiza la medida de rendimiento del agente *A*. Por tanto, el ajedrez es un entorno multiagente **competitivo**. Por otro lado, en el medio definido por el taxista circulando, el evitar colisiones maximiza la medida de rendimiento de todos los agentes, así pues es un entorno multiagente parcialmente **cooperativo**. Es también parcialmente competitivo ya que, por ejemplo, sólo un coche puede ocupar una plaza de aparcamiento. Los problemas en el diseño de agentes que aparecen en los entornos multiagente son a menudo bastante diferentes de los que aparecen en entornos con un único agente; por ejemplo, la **comunicación** a menudo emerge como un comportamiento racional en entornos multiagente; en algunos entornos competitivos parcialmente observables el **comportamiento estocástico** es racional ya que evita las dificultades de la predicción.

COMPETITIVO

COOPERATIVO

Como es de esperar, el caso más complejo es el *parcialmente observable*, *estocástico*, *secuencial*, *dinámico*, *continuo* y *multiagente*. De hecho, suele suceder que la mayoría de las situaciones reales son tan complejas que sería discutible clasificarlas como *realmente* deterministas. A efectos prácticos, se deben tratar como estocásticas. Un taxista circulando es un problema, complejo a todos los efectos.

La Figura 2.6 presenta las propiedades de un número de entornos familiares. Hay que tener en cuenta que las respuestas no están siempre preparadas de antemano. Por ejemplo, se ha presentado el ajedrez como totalmente observable; en sentido estricto, esto es falso porque ciertas reglas que afectan al movimiento de las torres, el enroque y a movimientos por repetición requieren que se recuerden algunos hechos sobre la historia del juego que no están reflejados en el estado del tablero. Estas excepciones, por supuesto, no tienen importancia si las comparamos con aquellas que aparecen en el caso del taxista, el tutor de inglés, o el sistema de diagnóstico médico.

Entornos de trabajo	Observable	Determinista	Episódico	Estático	Discreto	Agentes
Crucigrama	Totalmente	Determinista	Secuencial	Estático	Discreto	Individual
Ajedrez con reloj	Totalmente	Estratégico	Secuencial	Semi	Discreto	Multi
Póker	Parcialmente	Estratégico	Secuencial	Estático	Discreto	Multi
Backgammon	Totalmente	Estocástico	Secuencial	Estático	Discreto	Multi
Taxi circulando	Parcialmente	Estocástico	Secuencial	Dinámico	Continuo	Multi
Diagnóstico médico	Parcialmente	Estocástico	Secuencial	Dinámico	Continuo	Individual
Análisis de imagen	Totalmente	Determinista	Episódico	Semi	Continuo	Individual
Robot clasificador	Parcialmente	Estocástico	Episódico	Dinámico	Continuo	Individual
Controlador de refinería	Parcialmente	Estocástico	Secuencial	Dinámico	Continuo	Individual
Tutor interactivo de inglés	Parcialmente	Estocástico	Secuencial	Dinámico	Discreto	Multi

Figura 2.6 Ejemplos de entornos de trabajo y sus características.

Otras entradas de la tabla dependen de cómo se haya definido el entorno de trabajo. Se ha definido el sistema de diagnóstico médico como un único agente porque no es rentable modelar el proceso de la enfermedad en un paciente como un agente; pero incluso el sistema de diagnóstico médico podría necesitar tener en cuenta a pacientes recalcitrantes y empleados escépticos, de forma que el entorno podría tener un aspecto multiagente. Más aún, el diagnóstico médico es episódico si se concibe como proporcionar un diagnóstico a partir de una lista de síntomas; el problema es secuencial si ello trae consigo la propuesta de una serie de pruebas, un proceso de evaluación a lo largo del tratamiento, y demás aspectos. Muchos entornos son, también, episódicos si se observan desde un nivel de abstracción más alto que el de las acciones individuales del agente. Por ejemplo, un torneo de ajedrez consiste en una secuencia de juegos; cada juego es un episodio, pero (a la larga) la contribución de los movimientos en una partida al resultado general que obtenga el agente no se ve afectada por los movimientos realizados en la partida anterior. Por otro lado, las decisiones tomadas en una partida concreta son ciertamente de tipo secuencial.

El repositorio de código asociado a este libro (aima.cs.berkeley.edu) incluye la implementación de un número de entornos, junto con un simulador de entornos de propósito general que sitúa uno o más agentes en un entorno simulado, observa su comportamiento a lo largo del tiempo, y los evalúa de acuerdo con una medida de rendimiento dada. Estos experimentos no sólo se han realizado para un medio concreto, sino que se han realizado con varios problemas obtenidos de una **clase de entornos**. Por ejemplo, para evaluar un taxista en un tráfico simulado, sería interesante hacer varias simulaciones con diferente tipo de tráfico, claridad y condiciones atmosféricas. Si se diseña un agente para un escenario concreto, se pueden sacar ventajas de las propiedades específicas de ese caso en particular, pero puede no identificarse un buen diseño para conducir en general. Por esta razón, el repositorio de código también incluye un **generador de entornos** para cada clase de medios que selecciona hábitats particulares (con ciertas posibilidades) en los que ejecutar los agentes. Por ejemplo, el generador de un entorno

CLASE DE ENTORNOS

GENERADOR DE ENTORNOS

para un agente aspiradora inicializa el patrón de suciedad y la localización del agente de forma aleatoria. Después, es interesante evaluar la eficacia media del agente en el contexto de la clase del entorno. Un agente racional para una clase de entorno maximiza el rendimiento medio. Los Ejercicios del 2.7 al 2.12 guían el proceso de desarrollo de una clase de entornos y la evaluación de varios agentes.

2.4 Estructura de los agentes

PROGRAMA DEL AGENTE

ARQUITECTURA

Hasta este momento se ha hablado de los agentes describiendo su *conducta*, la acción que se realiza después de una secuencia de percepciones dada. Ahora, se trata de centrarse en el núcleo del problema y hablar sobre cómo trabajan internamente. El trabajo de la IA es diseñar el **programa del agente** que implemente la función del agente que proyecta las percepciones en las acciones. Se asume que este programa se ejecutará en algún tipo de computador con sensores físicos y actuadores, lo cual se conoce como **arquitectura:**

$$Agente = arquitectura + programa$$

Obviamente, el programa que se elija tiene que ser apropiado para la arquitectura. Si el programa tiene que recomendar acciones como *Caminar*, la arquitectura tiene que tener piernas. La arquitectura puede ser un PC común, o puede ser un coche robotizado con varios computadores, cámaras, y otros sensores a bordo. En general, la arquitectura hace que las percepciones de los sensores estén disponibles para el programa, ejecuta los programas, y se encarga de que los actuadores pongan en marcha las acciones generadas. La mayor parte de este libro se centra en el diseño de programas para agentes, aunque los Capítulos 24 y 25 tratan sobre sensores y actuadores.

Programas de los agentes

Los programas de los agentes que se describen en este libro tienen la misma estructura: reciben las percepciones actuales como entradas de los sensores y devuelven una acción a los actuadores[7]. Hay que tener en cuenta la diferencia entre los programas de los agentes, que toman la percepción actual como entrada, y la función del agente, que recibe la percepción histórica completa. Los programas de los agentes reciben sólo la percepción actual como entrada porque no hay nada más disponible en el entorno; si las acciones del agente dependen de la secuencia completa de percepciones, el agente tendría que recordar las percepciones.

Los programas de los agente se describirán con la ayuda de un sencillo lenguaje pseudocódigo que se define en el Apéndice B. El repositorio de código disponible en Inter-

[7] Hay otras posibilidades para definir la estructura del programa para el agente; por ejemplo, los programas para agentes pueden ser **subrutinas** que se ejecuten asincrónicamente en el entorno de trabajo. Cada una de estas subrutinas tienen un puerto de entrada y salida y consisten en un bucle que interpreta las entradas del puerto como percepciones y escribe acciones en el puerto de salida.

```
función AGENTE-DIRIGIDO-MEDIANTE TABLA(percepción) devuelve una acción
    variables estáticas: percepciones, una secuencia, vacía inicialmente
                         tabla, una tabla de acciones, indexada por las secuencias de
                         percepciones, totalmente definida inicialmente

    añadir la percepción al final de las percepciones
    acción ← CONSULTA(percepciones, tabla)
    devolver acción
```

Figura 2.7 El programa AGENTE-DIRIGIDO-MEDIANTE TABLA se invoca con cada nueva percepción y devuelve una acción en cada momento. Almacena la secuencia de percepciones utilizando su propia estructura de datos privada.

net contiene implementaciones en lenguajes de programación reales. Por ejemplo, la Figura 2.7 muestra un programa de agente muy sencillo que almacena la secuencia de percepciones y después las compara con las secuencias almacenadas en la tabla de acciones para decidir qué hacer. La tabla representa explícitamente la función que define el programa del agente. Para construir un agente racional de esta forma, los diseñadores deben realizar una tabla que contenga las acciones apropiadas para cada secuencia posible de percepciones.

Intuitivamente se puede apreciar por qué la propuesta de dirección-mediante-tabla para la construcción de agentes está condenada al fracaso. Sea P el conjunto de posibles percepciones y T el tiempo de vida del agente (el número total de percepciones que recibirá). La tabla de búsqueda contendrá $\sum_{t=1}^{T} |P|^t$ entradas. Si consideramos ahora el taxi automatizado: la entrada visual de una cámara individual es de 27 megabytes por segundo (30 fotografías por segundo, 640 $\times$ 480 pixels con 24 bits de información de colores). Lo cual genera una tabla de búsqueda con más de $10^{250.000.000.000}$ entradas por hora de conducción. Incluso la tabla de búsqueda del ajedrez (un fragmento del mundo real pequeño y obediente) tiene por lo menos 10^{150} entradas. El tamaño exageradamente grande de estas tablas (el número de átomos en el universo observable es menor que 10^{80}) significa que (a) no hay agente físico en este universo que tenga el espacio suficiente como para almacenar la tabla, (b) el diseñador no tendrá tiempo para crear la tabla, (c) ningún agente podría aprender todas las entradas de la tabla a partir de su experiencia, y (d) incluso si el entorno es lo suficientemente simple para generar una tabla de un tamaño razonable, el diseñador no tiene quien le asesore en la forma en la que rellenar la tabla.

A pesar de todo ello, el AGENTE-DIRIGIDO-MEDIANTE TABLA *hace* lo que nosotros queremos: implementa la función deseada para el agente. El desafío clave de la IA es encontrar la forma de escribir programas, que en la medida de lo posible, reproduzcan un comportamiento racional a partir de una pequeña cantidad de código en vez de a partir de una tabla con un gran número de entradas. Existen bastantes ejemplos que muestran qué se puede hacer con éxito en otras áreas: por ejemplo, las grandes tablas de las raíces cuadradas utilizadas por ingenieros y estudiantes antes de 1970 se han reemplazado por un programa de cinco líneas que implementa el método de Newton en las calculadoras electrónicas. La pregunta es, en el caso del comportamiento inteligente general, ¿puede la IA hacer lo que Newton hizo con las raíces cuadradas? Creemos que la respuesta es afirmativa.

En lo que resta de esta sección se presentan los cuatro tipos básicos de programas para agentes que encarnan los principios que subyacen en casi todos los sistemas inteligentes.

- Agentes reactivos simples.
- Agentes reactivos basados en modelos.
- Agentes basados en objetivos.
- Agentes basados en utilidad.

Después se explica, en términos generales, cómo convertir todos ellos en *agentes que aprendan*.

Agentes reactivos simples

AGENTE REACTIVO SIMPLE

El tipo de agente más sencillo es el **agente reactivo simple**. Estos agentes seleccionan las acciones sobre la base de las percepciones *actuales*, ignorando el resto de las percepciones históricas. Por ejemplo, el agente aspiradora cuya función de agente se presentó en la Figura 2.3 es un agente reactivo simple porque toma sus decisiones sólo con base en la localización actual y si ésta está sucia. La Figura 2.8 muestra el programa para este agente.

Hay que tener en cuenta que el programa para el agente aspiradora es muy pequeño comparado con su tabla correspondiente. La reducción más clara se obtiene al ignorar la historia de percepción, que reduce el número de posibilidades de 4^T a sólo 4. Otra reducción se basa en el hecho de que cuando la cuadrícula actual está sucia, la acción no depende de la localización.

Imagínese que es el conductor del taxi automático. Si el coche que circula delante frena, y las luces de freno se encienden, entonces lo advertiría y comenzaría a frenar. En otras palabras, se llevaría a cabo algún tipo de procesamiento sobre las señales visuales para establecer la condición que se llama «El coche que circula delante está frenando». Esto dispara algunas conexiones establecidas en el programa del agente para que se ejecute la acción «iniciar frenado». Esta conexión se denomina **regla de condición-acción**[8], y se representa por

REGLA DE CONDICIÓN-ACCIÓN

si *el-coche-que-circula-delante-está-frenando* **entonces** *iniciar-frenada*.

función AGENTE-ASPIRADORA-REACTIVO([*localización*, *estado*]) **devuelve** una acción

si *estado* = *Sucio* **entonces devolver** *Aspirar*
de otra forma, si *localización* = *A* **entonces devolver** *Derecha*
de otra forma, si *localización* = *B* **entonces devolver** *Izquierda*

Figura 2.8 Programa para el agente aspiradora de reactivo simple en el entorno definido por las dos cuadrículas. Este programa implementa la función de agente presentada en la Figura 2.3.

[8] También llamadas **reglas de situación-acción**, **producciones**, o **reglas si-entonces**.

Los humanos también tienen muchas de estas conexiones, algunas de las cuales son respuestas aprendidas (como en el caso de la conducción) y otras son reacciones innatas (como parpadear cuando algo se acerca al ojo). A lo largo de esta obra, se estudiarán diferentes formas en las que se pueden aprender e implementar estas conexiones.

El programa de la Figura 2.8 es específico para el entorno concreto de la aspiradora. Una aproximación más general y flexible es la de construir primero un intérprete de propósito general para reglas de condición-acción y después crear conjuntos de reglas para entornos de trabajo específicos. La Figura 2.9 presenta la estructura de este programa general de forma esquemática, mostrando cómo las reglas de condición-acción permiten al agente generar la conexión desde las percepciones a las acciones. No se preocupe si le parece trivial; pronto se complicará. Se utilizan rectángulos para denotar el estado interno actual del proceso de toma de decisiones del agente y óvalos para representar la información base utilizada en el proceso. El programa del agente, que es también muy simple, se muestra en la Figura 2.10. La función INTERPRETAR-ENTRADA genera una descripción abstracta del estado actual a partir de la percepción, y la función REGLA-COINCIDENCIA devuelve la primera regla del conjunto de reglas que coincide con la descripción del estado dada. Hay que tener en cuenta que la descripción en términos de «reglas»

Figura 2.9 Diagrama esquemático de un agente reactivo simple.

función AGENTE-REACTIVO-SIMPLE(percepción) **devuelve** una acción
  estático: *reglas*, un conjunto de reglas condición-acción

  estado ← INTERPRETAR-ENTRADA(*percepción*)
  regla ← REGLA-COINCIDENCIA(*estado, reglas*)
  acción ← REGLA-ACCIÓN[*regla*]
  devolver *acción*

Figura 2.10 Un agente reactivo simple, que actúa de acuerdo a la regla cuya condición coincida con el estado actual, definido por la percepción.

y «coincidencias» es puramente conceptual; las implementaciones reales pueden ser tan simples como colecciones de puertas lógicas implementando un circuito booleano.

Los agentes reactivos simples tienen la admirable propiedad de ser simples, pero poseen una inteligencia muy limitada. El agente de la Figura 2.10 funcionará *sólo si se puede tomar la decisión correcta sobre la base de la percepción actual, lo cual es posible sólo si el entorno es totalmente observable*. Incluso el que haya una pequeña parte que no se pueda observar puede causar serios problemas. Por ejemplo, la regla de frenado dada anteriormente asume que la condición *el-coche-que-circula-delante-está-frenando* se puede determinar a partir de la percepción actual (imagen de vídeo actual) si el coche de enfrente tiene un sistema centralizado de luces de freno. Desafortunadamente, los modelos antiguos tienen diferentes configuraciones de luces traseras, luces de frenado, y de intermitentes, y no es siempre posible saber a partir de una única imagen si el coche está frenando. Un agente reactivo simple conduciendo detrás de un coche de este tipo puede frenar continuamente y de manera innecesaria, o peor, no frenar nunca.

Un problema similar aparece en el mundo de la aspiradora. Supongamos que se elimina el sensor de localización de un agente aspiradora reactivo simple, y que sólo tiene un sensor de suciedad. Un agente de este tipo tiene sólo dos percepciones posibles: [*Sucio*] y [*Limpio*]. Puede *Aspirar* cuando se encuentra con [*Sucio*]. ¿Qué debe hacer cuando se encuentra con [*Limpio*]? Si se desplaza a la *Izquierda* se equivoca (siempre) si está en la cuadrícula *A*, y si de desplaza a la *Derecha* se equivoca (siempre) si está en la cuadrícula *B*. Los bucles infinitos son a menudo inevitables para los agentes reactivos simples que operan en algunos entornos parcialmente observables.

ALEATORIO

Salir de los bucles infinitos es posible si los agentes pueden seleccionar sus acciones **aleatoriamente**. Por ejemplo, si un agente aspiradora percibe [*Limpio*], puede lanzar una moneda y elegir entre *Izquierda* y *Derecha*. Es fácil mostrar que el agente se moverá a la otra cuadrícula en una media de dos pasos. Entonces, si la cuadrícula está sucia, la limpiará y la tarea de limpieza se completará. Por tanto, un agente reactivo simple con capacidad para elegir acciones de manera aleatoria puede mejorar los resultados que proporciona un agente reactivo simple determinista.

En la Sección 2.3 se mencionó que un comportamiento aleatorio de un tipo adecuado puede resultar racional en algunos entornos multiagente. En entornos de agentes individuales, el comportamiento aleatorio *no* es normalmente racional. Es un truco útil que ayuda a los agentes reactivos simples en algunas situaciones, pero en la mayoría de los casos se obtendrán mejores resultados con agentes deterministas más sofisticados.

Agentes reactivos basados en modelos

ESTADO INTERNO

La forma más efectiva que tienen los agentes de manejar la visibilidad parcial es *almacenar información de las partes del mundo que no pueden ver*. O lo que es lo mismo, el agente debe mantener algún tipo de **estado interno** que dependa de la historia percibida y que de ese modo refleje por lo menos alguno de los aspectos no observables del estado actual. Para el problema de los frenos, el estado interno no es demasiado extenso, sólo la fotografía anterior de la cámara, facilitando al agente la detección de dos luces rojas encendiéndose y apagándose simultáneamente a los costados del vehículo. Para

otros aspectos de la conducción, como un cambio de carril, el agente tiene que mantener información de la posición del resto de los coches si no los puede ver.

La actualización de la información de estado interno según pasa el tiempo requiere codificar dos tipos de conocimiento en el programa del agente. Primero, se necesita alguna información acerca de cómo evoluciona el mundo independientemente del agente, por ejemplo, que un coche que está adelantando estará más cerca, detrás, que en un momento inmediatamente anterior. Segundo, se necesita más información sobre cómo afectan al mundo las acciones del agente, por ejemplo, que cuando el agente gire hacia la derecha, el coche gira hacia la derecha o que después de conducir durante cinco minutos hacia el norte en la autopista se avanzan cinco millas hacia el norte a partir del punto en el que se estaba cinco minutos antes. Este conocimiento acerca de «cómo funciona el mundo», tanto si está implementado con un circuito booleano simple o con teorías científicas completas, se denomina **modelo** del mundo. Un agente que utilice este modelo es un **agente basado en modelos**.

AGENTE BASADO EN MODELOS

Figura 2.11 Un agente reactivo basado en modelos.

```
función AGENTE-REACTIVO-CON-ESTADO(percepción) devuelve una acción
    estático: estado, una descripción actual del estado del mundo
              reglas, un conjunto de reglas condición-acción
              acción, la acción más reciente, inicialmente ninguna

    estado ← ACTUALIZAR-ESTADO(estado, acción, percepción)
    regla ← REGLA-COINCIDENCIA(estado, reglas)
    acción ← REGLA-ACCIÓN[regla]
    devolver acción
```

Figura 2.12 Un agente reactivo basado en modelos, que almacena información sobre el estado actual del mundo utilizando un modelo interno. Después selecciona una acción de la misma forma que el agente reactivo.

La Figura 2.11 proporciona la estructura de un agente reactivo simple con estado interno, muestra cómo la percepción actual se combina con el estado interno antiguo para generar la descripción actualizada del estado actual. La Figura 2.12 muestra el programa del agente. La parte interesante es la correspondiente a la función Actualizar-Estado, que es la responsable de la creación de la nueva descripción del estado interno. Además de interpretar la nueva percepción a partir del conocimiento existente sobre el estado, utiliza información relativa a la forma en la que evoluciona el mundo para conocer más sobre las partes del mundo que no están visibles; para ello debe conocer cuál es el efecto de las acciones del agente sobre el estado del mundo. Los Capítulos 10 y 17 ofrecen ejemplos detallados.

Agentes basados en objetivos

El conocimiento sobre el estado actual del mundo no es siempre suficiente para decidir qué hacer. Por ejemplo, en un cruce de carreteras, el taxista puede girar a la izquierda, girar a la derecha o seguir hacia adelante. La decisión correcta depende de dónde quiere ir el taxi. En otras palabras, además de la descripción del estado actual, el agente necesita algún tipo de información sobre su **meta** que describa las situaciones que son deseables, por ejemplo, llegar al destino propuesto por el pasajero. El programa del agente se puede combinar con información sobre los resultados de las acciones posibles (la misma información que se utilizó para actualizar el estado interno en el caso del agente reflexivo) para elegir las acciones que permitan alcanzar el objetivo. La Figura 2.13 muestra la estructura del agente basado en objetivos.

META

En algunas ocasiones, la selección de acciones basadas en objetivos es directa, cuando alcanzar los objetivos es el resultado inmediato de una acción individual. En otras oca-

Figura 2.13 Un agente basado en objetivos y basado en modelos, que almacena información del estado del mundo así como del conjunto de objetivos que intenta alcanzar, y que es capaz de seleccionar la acción que eventualmente lo guiará hacia la consecución de sus objetivos.

siones, puede ser más complicado, cuando el agente tiene que considerar secuencias complejas para encontrar el camino que le permita alcanzar el objetivo. **Búsqueda** (Capítulos del 3 al 6) y **planificación** (Capítulos 11 y 12) son los subcampos de la IA centrados en encontrar secuencias de acciones que permitan a los agentes alcanzar sus metas.

Hay que tener en cuenta que la toma de decisiones de este tipo es fundamentalmente diferente de las reglas de condición–acción descritas anteriormente, en las que hay que tener en cuenta consideraciones sobre el futuro (como «¿qué pasará si yo hago esto y esto?» y «¿me hará esto feliz?»). En los diseños de agentes reactivos, esta información no está representada explícitamente, porque las reglas que maneja el agente proyectan directamente las percepciones en las acciones. El agente reactivo frena cuando ve luces de freno. Un agente basado en objetivos, en principio, puede razonar que si el coche que va delante tiene encendidas las luces de frenado, está reduciendo su velocidad. Dada la forma en la que el mundo evoluciona normalmente, la única acción que permite alcanzar la meta de no chocarse con otros coches, es frenar.

Aunque el agente basado en objetivos pueda parecer menos eficiente, es más flexible ya que el conocimiento que soporta su decisión está representado explícitamente y puede modificarse. Si comienza a llover, el agente puede actualizar su conocimiento sobre cómo se comportan los frenos; lo cual implicará que todas las formas de actuar relevantes se alteren automáticamente para adaptarse a las nuevas circunstancias. Para el agente reactivo, por otro lado, se tendrán que rescribir muchas reglas de condición-acción. El comportamiento del agente basado en objetivos puede cambiarse fácilmente para que se dirija a una localización diferente. Las reglas de los agentes reactivos relacionadas con cuándo girar y cuándo seguir recto son válidas sólo para un destino concreto y tienen que modificarse cada vez que el agente se dirija a cualquier otro lugar distinto.

Agentes basados en utilidad

Las metas por sí solas no son realmente suficientes para generar comportamiento de gran calidad en la mayoría de los entornos. Por ejemplo, hay muchas secuencias de acciones que llevarán al taxi a su destino (y por tanto a alcanzar su objetivo), pero algunas son más rápidas, más seguras, más fiables, o más baratas que otras. Las metas sólo proporcionan una cruda distinción binaria entre los estados de «felicidad» y «tristeza», mientras que una medida de eficiencia más general debería permitir una comparación entre estados del mundo diferentes de acuerdo al nivel exacto de felicidad que el agente alcance cuando se llegue a un estado u otro. Como el término «felicidad» no suena muy científico, la terminología tradicional utilizada en estos casos para indicar que se prefiere un estado del mundo a otro es que un estado tiene más **utilidad** que otro para el agente[9].

UTILIDAD

FUNCIÓN DE UTILIDAD

Una **función de utilidad** proyecta un estado (o una secuencia de estados) en un número real, que representa un nivel de felicidad. La definición completa de una función de utilidad permite tomar decisiones racionales en dos tipos de casos en los que las metas son inadecuadas. Primero, cuando haya objetivos conflictivos, y sólo se puedan al-

[9] La palabra «utilidad» aquí se refiere a «la cualidad de ser útil».

Figura 2.14 Un agente basado en utilidad y basado en modelos. Utiliza un modelo del mundo, junto con una función de utilidad que calcula sus preferencias entre los estados del mundo. Después selecciona la acción que le lleve a alcanzar la mayor utilidad esperada, que se calcula haciendo la media de todos los estados resultantes posibles, ponderado con la probabilidad del resultado.

canzar algunos de ellos (por ejemplo, velocidad y seguridad), la función de utilidad determina el equilibrio adecuado. Segundo, cuando haya varios objetivos por los que se pueda guiar el agente, y ninguno de ellos se pueda alcanzar con certeza, la utilidad proporciona un mecanismo para ponderar la probabilidad de éxito en función de la importancia de los objetivos.

En el Capítulo 16, se mostrará cómo cualquier agente racional debe comportarse *como si* tuviese una función de utilidad cuyo valor esperado tiene que maximizar. Por tanto, un agente que posea una función de utilidad *explícita* puede tomar decisiones racionales, y lo puede hacer con la ayuda de un algoritmo de propósito general que no dependa de la función específica de utilidad a maximizar. De esta forma, la definición «global» de racionalidad (identificando como racionales aquellas funciones de los agentes que proporcionan el mayor rendimiento) se transforma en una restricción «local» en el diseño de agentes racionales que se puede expresar con un simple programa.

La Figura 2.14 muestra la estructura de un agente basado en utilidad. En la Parte IV aparecen programas de agentes basados en utilidad, donde se presentan agentes que toman decisiones y que deben trabajar con la incertidumbre inherente a los entornos parcialmente observables.

Agentes que aprenden

Se han descrito programas para agentes que poseen varios métodos para seleccionar acciones. Hasta ahora no se ha explicado cómo *poner en marcha* estos programas de agentes. Turing (1950), en su temprano y famoso artículo, consideró la idea de programar sus máquinas inteligentes a mano. Estimó cuánto tiempo podía llevar y concluyó que «Se-

ría deseable utilizar algún método más rápido». El método que propone es construir máquinas que aprendan y después enseñarlas. En muchas áreas de IA, éste es ahora el método más adecuado para crear sistemas novedosos. El aprendizaje tiene otras ventajas, como se ha explicado anteriormente: permite que el agente opere en medios inicialmente desconocidos y que sea más competente que si sólo utilizase un conocimiento inicial. En esta sección, se introducen brevemente las principales ideas en las que se basan los agentes que aprenden. En casi todos los capítulos de este libro se comentan las posibilidades y métodos de aprendizaje de tipos de agentes concretos. La Parte VI profundiza más en los algoritmos de aprendizaje en sí mismos.

ELEMENTO DE APRENDIZAJE

ELEMENTO DE ACTUACIÓN

CRÍTICA

Un agente que aprende se puede dividir en cuatro componentes conceptuales, tal y como se muestra en la Figura 2.15. La distinción más importante entre el **elemento de aprendizaje** y el **elemento de actuación** es que el primero está responsabilizado de hacer mejoras y el segundo se responsabiliza de la selección de acciones externas. El elemento de actuación es lo que anteriormente se había considerado como el agente completo: recibe estímulos y determina las acciones a realizar. El elemento de aprendizaje se realimenta con las **críticas** sobre la actuación del agente y determina cómo se debe modificar el elemento de actuación para proporcionar mejores resultados en el futuro.

El diseño del elemento de aprendizaje depende mucho del diseño del elemento de actuación. Cuando se intenta diseñar un agente que tenga capacidad de aprender, la primera cuestión a solucionar no es ¿cómo se puede enseñar a aprender?, sino ¿qué tipo de elemento de actuación necesita el agente para llevar a cabo su objetivo, cuando haya aprendido cómo hacerlo? Dado un diseño para un agente, se pueden construir los mecanismos de aprendizaje necesarios para mejorar cada una de las partes del agente.

La crítica indica al elemento de aprendizaje qué tal lo está haciendo el agente con respecto a un nivel de actuación fijo. La crítica es necesaria porque las percepciones por sí mismas no prevén una indicación del éxito del agente. Por ejemplo, un programa de

Figura 2.15 Modelo general para agentes que aprenden.

ajedrez puede recibir una percepción indicando que ha dado jaque mate a su oponente, pero necesita tener un nivel de actuación que le indique que ello es bueno; la percepción por sí misma no lo indica. Es por tanto muy importante fijar el nivel de actuación. Conceptualmente, se debe tratar con él como si estuviese fuera del agente, ya que éste no debe modificarlo para satisfacer su propio interés.

GENERADOR DE PROBLEMAS

El último componente del agente con capacidad de aprendizaje es el **generador de problemas**. Es responsable de sugerir acciones que lo guiarán hacia experiencias nuevas e informativas. Lo interesante es que si el elemento de actuación sigue su camino, puede continuar llevando a cabo las acciones que sean mejores, dado su conocimiento. Pero si el agente está dispuesto a explorar un poco, y llevar a cabo algunas acciones que no sean totalmente óptimas a corto plazo, puede descubrir acciones mejores a largo plazo. El trabajo del generador de problemas es sugerir estas acciones exploratorias. Esto es lo que los científicos hacen cuando llevan a cabo experimentos. Galileo no pensaba que tirar piedras desde lo alto de una torre en Pisa tenía un valor por sí mismo. Él no trataba de romper piedras ni de cambiar la forma de pensar de transeúntes desafortunados que paseaban por el lugar. Su intención era adaptar su propia mente, para identificar una teoría que definiese mejor el movimiento de los objetos.

Para concretar el diseño total, se puede volver a utilizar el ejemplo del taxi automatizado. El elemento de actuación consiste en la colección de conocimientos y procedimientos que tiene el taxi para seleccionar sus acciones de conducción. El taxi se pone en marcha y circula utilizando este elemento de actuación. La crítica observa el mundo y proporciona información al elemento de aprendizaje. Por ejemplo, después de que el taxi se sitúe tres carriles hacia la izquierda de forma rápida, la crítica observa el lenguaje escandaloso que utilizan otros conductores. A partir de esta experiencia, el elemento de aprendizaje es capaz de formular una regla que indica que ésta fue una mala acción, y el elemento de actuación se modifica incorporando la nueva regla. El generador de problemas debe identificar ciertas áreas de comportamiento que deban mejorarse y sugerir experimentos, como probar los frenos en carreteras con tipos diferentes de superficies y bajo condiciones distintas.

El elemento de aprendizaje puede hacer cambios en cualquiera de los componentes de «conocimiento» que se muestran en los diagramas de agente (Figuras 2.9, 2.11, 2.13, y 2.14). Los casos más simples incluyen el aprendizaje directo a partir de la secuencia percibida. La observación de pares de estados sucesivos del entorno puede permitir que el agente aprenda «cómo evoluciona el mundo», y la observación de los resultados de sus acciones puede permitir que el agente aprenda «qué hacen sus acciones». Por ejemplo, si el taxi ejerce una cierta presión sobre los frenos cuando está circulando por una carretera mojada, acto seguido conocerá cómo decelera el coche. Claramente, estas dos tareas de aprendizaje son más difíciles si sólo existe una vista parcial del medio.

Las formas de aprendizaje mostradas en los párrafos precedentes no necesitan el acceso a niveles de actuación externo, de alguna forma, el nivel es el que se utiliza universalmente para hacer pronósticos de acuerdo con la experimentación. La situación es ligeramente más compleja para un agente basado en utilidad que desee adquirir información para crear su función de utilidad. Por ejemplo, se supone que el agente conductor del taxi no recibe propina de los pasajeros que han recorrido un trayecto de forma incómoda debido a una mala conducción. El nivel de actuación externo debe informar

al agente de que la pérdida de propinas tiene una contribución negativa en su nivel de actuación medio; entonces el agente puede aprender que «maniobras violentas no contribuyen a su propia utilidad». De alguna manera, el nivel de actuación identifica parte de las percepciones entrantes como **recompensas** (o **penalizaciones**) que generan una respuesta directa en la calidad del comportamiento del agente. Niveles de actuación integrados como el dolor y el hambre en animales se pueden enmarcar en este contexto. El Capítulo 21 discute estos asuntos.

En resumen, los agentes tienen una gran variedad de componentes, y estos componentes se pueden representar de muchas formas en los programas de agentes, por lo que, parece haber una gran variedad de métodos de aprendizaje. Existe, sin embargo, una visión unificada sobre un tema fundamental. El aprendizaje en el campo de los agentes inteligentes puede definirse como el proceso de modificación de cada componente del agente, lo cual permite a cada componente comportarse más en consonancia con la información que se recibe, lo que por tanto permite mejorar el nivel medio de actuación del agente.

2.5 Resumen

En este capítulo se ha realizado un recorrido rápido por el campo de la IA, que se ha presentado como la ciencia del diseño de los agentes. Los puntos más importantes a tener en cuenta son:

- Un **agente** es algo que percibe y actúa en un medio. La **función del agente** para un agente especifica la acción que debe realizar un agente como respuesta a cualquier secuencia percibida.
- La **medida de rendimiento** evalúa el comportamiento del agente en un medio. Un **agente racional** actúa con la intención de maximizar el valor esperado de la medida de rendimiento, dada la secuencia de percepciones que ha observado hasta el momento.
- Las especificaciones del **entorno de trabajo** incluyen la medida de rendimiento, el medio externo, los actuadores y los sensores. El primer paso en el diseño de un agente debe ser siempre la especificación, tan completa como sea posible, del entorno de trabajo.
- El entorno de trabajo varía según distintos parámetros. Pueden ser total o parcialmente visibles, deterministas o estocásticos, episódicos o secuenciales, estáticos o dinámicos, discretos o continuos, y formados por un único agente o por varios agentes.
- El **programa del agente** implementa la función del agente. Existe una gran variedad de diseños de programas de agentes, y reflejan el tipo de información que se hace explícita y se utiliza en el proceso de decisión. Los diseños varían en eficiencia, solidez y flexibilidad. El diseño apropiado del programa del agente depende en gran medida de la naturaleza del medio.
- **Los gentes reactivos simples** responden directamente a las percepciones, mientras que los **agentes reactivos basados en modelos** mantienen un estado interno

que les permite seguir el rastro de aspectos del mundo que no son evidentes según las percepciones actuales. Los agentes basados en objetivos actúan con la intención de alcanzar sus metas, y los agentes basados en utilidad intentan maximizar su «felicidad» deseada.

- Todos los agentes pueden mejorar su eficacia con la ayuda de mecanismos de **aprendizaje**.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

El papel central de la acción en la inteligencia (la noción del razonamiento práctico) se remonta por lo menos a la obra *Nicomachean Ethics* de Aristóteles. McCarthy (1958) trató también el tema del razonamiento práctico en su influyente artículo *Programs with Common Sense*. Los campos de la robótica y la teoría de control tienen interés, por su propia naturaleza, en la construcción de agentes físicos. El concepto de un **controlador**, en el ámbito de la teoría de control, es idéntico al de un agente en IA. Quizá sorprendentemente, la IA se ha concentrado durante la mayor parte de su historia en componentes aislados de agentes (sistemas que responden a preguntas, demostración de teoremas, sistemas de visión, y demás) en vez de en agentes completos. La discusión sobre agentes que se presenta en el libro de Genesereth y Nilsson (1987) fue una influyente excepción. El concepto de agente en sí está aceptado ampliamente ahora en el campo y es un tema central en libros recientes (Poole *et al*., 1998; Nilsson, 1998).

CONTROLADOR

El Capítulo 1 muestra las raíces del concepto de racionalidad en la Filosofía y la Economía. En la IA, el concepto tuvo un interés periférico hasta mediados de los 80, donde comenzó a suscitar muchas discusiones sobre los propios fundamentos técnicos del campo. Un artículo de Jon Doyle (1983) predijo que el diseño de agentes racionales podría llegar a ser la misión central de la IA, mientras otras áreas populares podrían separarse dando lugar a nuevas disciplinas.

Es muy importante tener muy en cuenta las propiedades del medio y sus consecuencias cuando se realiza el diseño de los agentes racionales ya que forma parte de la tradición ligada a la teoría de control [por ejemplo los sistemas de control clásicos (Dorf y Bishop, 1999) manejan medios deterministas y totalmente observables; el control óptimo estocástico (Kumar y Varaiya, 1986) maneja medios parcialmente observables y estocásticos y un control híbrido (Henzinger y Sastry, 1998) maneja entornos que contienen elementos discretos y continuos]. La distinción entre entornos totalmente y parcialmente observables es también central en la literatura sobre **programación dinámica** desarrollada en el campo de la investigación operativa (Puterman, 1994), como se comentará en el Capítulo 17.

Los agentes reactivos fueron los primeros modelos para psicólogos conductistas como Skinner (1953), que intentó reducir la psicología de los organismos estrictamente a correspondencias entrada/salida o estímulo/respuesta. La evolución del behaviourismo hacia el funcionalismo en el campo de la psicología, que estuvo, al menos de forma parcial, dirigida por la aplicación de la metáfora del computador a los agentes (Putnam, 1960; Lewis, 1966) introdujo el estado interno del agente en el nuevo escenario. La mayor par-

te del trabajo realizado en el campo de la IA considera que los agentes reactivos puros con estado interno son demasiado simples para ser muy influyentes, pero los trabajos de Rosenschein (1985) y Brooks (1986) cuestionan esta hipótesis (*véase* el Capítulo 25). En los últimos años, se ha trabajado intensamente para encontrar algoritmos eficientes capaces de hacer un buen seguimiento de entornos complejos (Hamscher *et al*., 1992). El programa del Agente Remoto que controla la nave espacial Deep Space One (descrito en la página 27) es un admirable ejemplo concreto (Muscettola *et al*., 1998; Jonsson *et al*., 2000).

Los agentes basados en objetivos están presentes tanto en las referencias de Aristóteles sobre el razonamiento práctico como en los primeros artículos de McCarthy sobre IA lógica. El robot Shakey (Fikes y Nilsson, 1971; Nilsson, 1984) fue el primer robot construido como un agente basado en objetivos. El análisis lógico completo de un agente basado en objetivos aparece en Genesereth y Nilsson (1987), y Shoham (1993) ha desarrollado una metodología de programación basada en objetivos llamada programación orientada a agentes.

La perspectiva orientada a objetivos también predomina en la psicología cognitiva tradicional, concretamente en el área de la resolución de problemas, como se muestra tanto en el influyente *Human Problem Solving* (Newell y Simon, 1972) como en los últimos trabajos de Newell (1990). Los objetivos, posteriormente definidos como *deseos* (generales) y las *intenciones* (perseguidas en un momento dado), son fundamentales en la teoría de agentes desarrollada por Bratman (1987). Esta teoría ha sido muy influyente tanto en el entendimiento del lenguaje natural como en los sistemas multiagente.

Horvitz *et al*. (1988) sugieren específicamente el uso de la maximización de la utilidad esperada concebida racionalmente como la base de la IA. El texto de Pearl (1988) fue el primero en IA que cubrió las teorías de la probabilidad y la utilidad en profundidad; su exposición de métodos prácticos de razonamiento y toma de decisiones con incertidumbre fue, posiblemente, el factor individual que más influyó en el desarrollo de los agentes basados en utilidad en los 90 (*véase* la Parte V).

El diseño general de agentes que aprenden representado en la Figura 2.15 es un clásico de la literatura sobre aprendizaje automático (Buchanan *et al*., 1978; Mitchell, 1997). Ejemplos de diseños, implementados en programas, se remontan, como poco, hasta los programas que aprendían a jugar al ajedrez de Arthur Samuel (1959, 1967). La Parte VI está dedicada al estudio en profundidad de los agentes que aprenden.

El interés en los agentes y en el diseño de agentes ha crecido rápidamente en los últimos años, en parte por la expansión de Internet y la necesidad observada de desarrollar **softbots** (robots *software*) automáticos y móviles (Etzioni y Weld, 1994). Artículos relevantes pueden encontrarse en *Readings in Agents* (Huhns y Singh, 1998) y *Fundations of Rational Agency* (Wooldridge y Rao, 1999). *Multiagent Systems* (Weiss, 1999) proporciona una base sólida para muchos aspectos del diseño de agentes. Conferencias dedicadas a agentes incluyen la International Conference on Autonomous Agents, la International Workshop on Agent Theories, Architectures, and Languages, y la International Conference on Multiagent Systems. Finalmente, *Dung Beetle Ecology* (Hanski y Cambefort, 1991) proporciona gran cantidad de información interesante sobre el comportamiento de los escarabajos estercoleros.

EJERCICIOS

2.1 Defina con sus propias palabras los siguientes términos: agente, función de agente, programa de agente, racionalidad, autonomía, agente reactivo, agente basado en modelo, agente basado en objetivos, agente basado en utilidad, agente que aprende.

2.2 Tanto la medida de rendimiento como la función de utilidad miden la eficiencia del agente. Explique la diferencia entre los dos conceptos.

2.3 Este ejercicio explora las diferencias entre las funciones de los agentes y los programas de los agentes.

- ***a***) ¿Puede haber más de un programa de agente que implemente una función de agente dada? Proponga un ejemplo, o muestre por qué una no es posible.
- ***b***) ¿Hay funciones de agente que no se pueden implementar con algún programa de agente?
- ***c***) Dada una arquitectura máquina, ¿implementa cada programa de agente exactamente una función de agente?
- ***d***) Dada una arquitectura con n bits de almacenamiento, ¿cuántos posibles programas de agente diferentes puede almacenar?

2.4 Examínese ahora la racionalidad de varias funciones de agentes aspiradora.

- ***a***) Muestre que la función de agente aspiradora descrita en la Figura 2.3 es realmente racional bajo la hipótesis presentada en la página 36.
- ***b***) Describa una función para un agente racional cuya medida de rendimiento modificada deduzca un punto por cada movimiento. ¿Requiere el correspondiente programa de agente estado interno?
- ***c***) Discuta posibles diseños de agentes para los casos en los que las cuadrículas limpias puedan ensuciarse y la geografía del medio sea desconocida. ¿Tiene sentido que el agente aprenda de su experiencia en estos casos? ¿Si es así, qué debe aprender?

2.5 Identifique la descripción REAS que define el entorno de trabajo para cada uno de los siguientes agentes:

- ***a***) Robot que juega al fútbol;
- ***b***) Agente para comprar libros en Internet;
- ***c***) Explorador autónomo de Marte;
- ***d***) Asistente matemático para la demostración de teoremas.

2.6 Para cada uno de los tipos de agente enumerados en el Ejercicio 2.5, caracterice el medio de acuerdo con las propiedades dadas en la Sección 2.3, y seleccione un diseño de agente adecuado.

Los siguientes ejercicios están relacionados con la implementación de entornos y agentes para el mundo de la aspiradora.

2.7 Implemente un simulador que determine la medida de rendimiento para el entorno del mundo de la aspiradora descrito en la Figura 2.2 y especificado en la página 36. La implementación debe ser modular, de forma que los sensores, actuadores, y las características del entorno (tamaño, forma, localización de la suciedad, etc.) puedan modificar-

se fácilmente. (*Nota:* hay implementaciones disponibles en el repositorio de Internet que pueden ayudar a decidir qué lenguaje de programación y sistema operativo seleccionar).

2.8 Implemente un agente reactivo simple para el entorno de la aspiradora del Ejercicio 2.7. Ejecute el simulador del entorno con este agente para todas las configuraciones iniciales posibles de suciedad y posiciones del agente. Almacene la puntuación de la actuación del agente para cada configuración y la puntuación media global.

2.9 Considere una versión modificada del entorno de la aspiradora del Ejercicio 2.7, en el que se penalice al agente con un punto en cada movimiento.

- ***a***) ¿Puede un agente reactivo simple ser perfectamente racional en este medio? Explíquese.
- ***b***) ¿Qué sucedería con un agente reactivo con estado? Diseñe este agente.
- ***c***) ¿Cómo se responderían las preguntas **a** y **b** si las percepciones proporcionan al agente información sobre el nivel de suciedad/limpieza de todas las cuadrículas del entorno?

2.10 Considere una versión modificada del entorno de la aspiradora del Ejercicio 2.7, en el que la geografía del entorno (su extensión, límites, y obstáculos) sea desconocida, así como, la disposición inicial de la suciedad. (El agente puede ir hacia *arriba*, *abajo,* así como, hacia la *derecha* y a la *izquierda*.)

- ***a***) ¿Puede un agente reactivo simple ser perfectamente racional en este medio? Explíquese.
- ***b***) ¿Puede un agente reactivo simple con una función de agente aleatoria superar a un agente reactivo simple? Diseñe un agente de este tipo y medir su rendimiento en varios medios.
- ***c***) ¿Se puede diseñar un entorno en el que el agente con la función aleatoria obtenga una actuación muy pobre? Muestre los resultados.
- ***d***) ¿Puede un agente reactivo con estado mejorar los resultados de un agente reactivo simple? Diseñe un agente de este tipo y medir su eficiencia en distintos medios. ¿Se puede diseñar un agente racional de este tipo?

2.11 Repítase el Ejercicio 2.10 para el caso en el que el sensor de localización sea reemplazado por un sensor «de golpes» que detecte si el agente golpea un obstáculo o si se sale fuera de los límites del entorno. Supóngase que el sensor de golpes deja de funcionar. ¿Cómo debe comportarse el agente?

2.12 Los entornos de la aspiradora en los ejercicios anteriores han sido todos deterministas. Discuta posibles programas de agentes para cada una de las siguientes versiones estocásticas:

- ***a***) Ley de Murphy: el 25 por ciento del tiempo, la acción de *Aspirar* falla en la limpieza del suelo si está sucio y deposita suciedad en el suelo si el suelo está limpio. ¿Cómo se ve afectado el agente si el sensor de suciedad da una respuesta incorrecta el diez por ciento de las veces?
- ***b***) Niño pequeño: en cada lapso de tiempo, cada recuadro limpio tiene un diez por ciento de posibilidad de ensuciarse. ¿Puede identificar un diseño para un agente racional en este caso?

3 Resolver problemas mediante búsqueda

En donde veremos cómo un agente puede encontrar una secuencia de acciones que alcance sus objetivos, cuando ninguna acción simple lo hará.

Los agentes más simples discutidos en el Capítulo 2 fueron los agentes reactivos, los cuales basan sus acciones en una aplicación directa desde los estados a las acciones. Tales agentes no pueden funcionar bien en entornos en los que esta aplicación sea demasiado grande para almacenarla y que tarde mucho en aprenderla. Por otra parte, los agentes basados en objetivos pueden tener éxito considerando las acciones futuras y lo deseable de sus resultados.

AGENTE RESOLVENTE-PROBLEMAS

Este capítulo describe una clase de agente basado en objetivos llamado **agente resolvente-problemas**. Los agentes resolventes-problemas deciden qué hacer para encontrar secuencias de acciones que conduzcan a los estados deseables. Comenzamos definiendo con precisión los elementos que constituyen el «problema» y su «solución», y daremos diferentes ejemplos para ilustrar estas definiciones. Entonces, describimos diferentes algoritmos de propósito general que podamos utilizar para resolver estos problemas y así comparar las ventajas de cada algoritmo. Los algoritmos son **no informados**, en el sentido que no dan información sobre el problema salvo su definición. El Capítulo 4 se ocupa de los algoritmos de búsqueda **informada**, los que tengan cierta idea de dónde buscar las soluciones.

Este capítulo utiliza los conceptos de análisis de algoritmos. Los lectores no familiarizados con los conceptos de complejidad asintótica (es decir, notación $O()$) y la NP completitud, debería consultar el Apéndice A.

3.1 Agentes resolventes-problemas

Se supone que los agentes inteligentes deben maximizar su medida de rendimiento. Como mencionamos en el Capítulo 2, esto puede simplificarse algunas veces si el agente puede

elegir **un objetivo** y trata de satisfacerlo. Primero miraremos el porqué y cómo puede hacerlo.

Imagine un agente en la ciudad de Arad, Rumanía, disfrutando de un viaje de vacaciones. La medida de rendimiento del agente contiene muchos factores: desea mejorar su bronceado, mejorar su rumano, tomar fotos, disfrutar de la vida nocturna, evitar resacas, etcétera. El problema de decisión es complejo implicando muchos elementos y por eso, lee cuidadosamente las guías de viaje. Ahora, supongamos que el agente tiene un billete no reembolsable para volar a Bucarest al día siguiente. En este caso, tiene sentido que el agente elija **el objetivo** de conseguir Bucarest. Las acciones que no alcanzan Bucarest se pueden rechazar sin más consideraciones y el problema de decisión del agente se simplifica enormemente. Los objetivos ayudan a organizar su comportamiento limitando las metas que intenta alcanzar el agente. El primer paso para solucionar un problema es la **formulación del objetivo**, basado en la situación actual y la medida de rendimiento del agente.

FORMULACIÓN DEL OBJETIVO

Consideraremos un objetivo como un conjunto de estados del mundo (exactamente aquellos estados que satisfacen el objetivo). La tarea del agente es encontrar qué secuencia de acciones permite obtener un estado objetivo. Para esto, necesitamos decidir qué acciones y estados considerar. Si se utilizaran acciones del tipo «mueve el pie izquierdo hacia delante» o «gira el volante un grado a la izquierda», probablemente el agente nunca encontraría la salida del aparcamiento, no digamos por tanto llegar a Bucarest, porque a ese nivel de detalle existe demasiada incertidumbre en el mundo y habría demasiados pasos en una solución. Dado un objetivo, la **formulación del problema** es el proceso de decidir qué acciones y estados tenemos que considerar. Discutiremos con más detalle este proceso. Por ahora, suponemos que el agente considerará acciones del tipo conducir de una ciudad grande a otra. Consideraremos los estados que corresponden a estar en una ciudad[1] determinada.

FORMULACIÓN DEL PROBLEMA

Ahora, nuestro agente ha adoptado el objetivo de conducir a Bucarest, y considera a dónde ir desde Arad. Existen tres carreteras desde Arad, una hacia Sibiu, una a Timisoara, y una a Zerind. Ninguna de éstas alcanza el objetivo, y, a menos que el agente este familiarizado con la geografía de Rumanía, no sabría qué carretera seguir[2]. En otras palabras, el agente no sabrá cuál de las posibles acciones es mejor, porque no conoce lo suficiente los estados que resultan al tomar cada acción. Si el agente no tiene conocimiento adicional, entonces estará en un callejón sin salida. Lo mejor que puede hacer es escoger al azar una de las acciones.

Pero, supongamos que el agente tiene un mapa de Rumanía, en papel o en su memoria. El propósito del mapa es dotar al agente de información sobre los estados en los que podría encontrarse, así como las acciones que puede tomar. El agente puede usar esta información para considerar los siguientes estados de un viaje hipotético por cada una de las tres ciudades, intentando encontrar un viaje que llegue a Bucarest. Una vez que

[1] Observe que cada uno de estos «estados» se corresponde realmente a un conjunto de estados del mundo, porque un estado del mundo real especifica todos los aspectos de realidad. Es importante mantener en mente la distinción entre estados del problema a resolver y los estados del mundo.

[2] Suponemos que la mayoría de los lectores están en la misma situación y pueden fácilmente imaginarse cómo de desorientado está nuestro agente. Pedimos disculpas a los lectores rumanos quienes no pueden aprovecharse de este recurso pedagógico.

ha encontrado un camino en el mapa desde Arad a Bucarest, puede alcanzar su objetivo, tomando las acciones de conducir que corresponden con los tramos del viaje. En general, *un agente con distintas opciones inmediatas de valores desconocidos puede decidir qué hacer, examinando las diferentes secuencias posibles de acciones que le conduzcan a estados de valores conocidos, y entonces escoger la mejor secuencia.*

BÚSQUEDA

SOLUCIÓN

EJECUCIÓN

Este proceso de hallar esta secuencia se llama **búsqueda**. Un algoritmo de búsqueda toma como entrada un problema y devuelve una **solución** de la forma secuencia de acciones. Una vez que encontramos una solución, se procede a ejecutar las acciones que ésta recomienda. Esta es la llamada fase de **ejecución**. Así, tenemos un diseño simple de un agente «formular, buscar, ejecutar», como se muestra en la Figura 3.1. Después de formular un objetivo y un problema a resolver, el agente llama al procedimiento de búsqueda para resolverlo. Entonces, usa la solución para guiar sus acciones, haciendo lo que la solución le indica como siguiente paso a hacer —generalmente, primera acción de la secuencia— y procede a eliminar este paso de la secuencia. Una vez ejecutada la solución, el agente formula un nuevo objetivo.

Primero describimos el proceso de formulación del problema, y después dedicaremos la última parte del capítulo a diversos algoritmos para la función BÚSQUEDA. En este capítulo no discutiremos las funciones ACTUALIZAR-ESTADO y FORMULAR-OBJETIVO.

Antes de entrar en detalles, hagamos una breve pausa para ver dónde encajan los agentes resolventes de problemas en la discusión de agentes y entornos del Capítulo 2. El agente diseñado en la Figura 3.1 supone que el entorno es **estático**, porque la formulación y búsqueda del problema se hace sin prestar atención a cualquier cambio que puede ocurrir en el entorno. El agente diseñado también supone que se conoce el estado inicial; conocerlo es fácil si el entorno es **observable**. La idea de enumerar «las lí-

```
función AGENTE-SENCILLO-RESOLVENTE-PROBLEMAS(percepción) devuelve una acción
  entradas: percepción, una percepción
  estático: sec, una secuencia de acciones, vacía inicialmente
            estado, una descripción del estado actual del mundo
            objetivo, un objetivo, inicialmente nulo
            problema, una formulación del problema

  estado ← ACTUALIZAR-ESTADO(estado, percepción)
  si sec está vacía entonces hacer
    objetivo ← FORMULAR-OBJETIVO(estado)
    problema ← FORMULAR-PROBLEMA(estado,objetivo)
    sec ← BÚSQUEDA(problema)
  acción ← PRIMERO(secuencia)
  sec ← RESTO(secuencia)
  devolver acción
```

Figura 3.1 Un sencillo agente resolvente de problemas. Primero formula un objetivo y un problema, busca una secuencia de acciones que deberían resolver el problema, y entonces ejecuta las acciones una cada vez. Cuando se ha completado, formula otro objetivo y comienza de nuevo. Notemos que cuando se ejecuta la secuencia, se ignoran sus percepciones: se supone que la solución encontrada trabajará bien.

neas de acción alternativas» supone que el entorno puede verse como **discreto**. Finalmente, y más importante, el agente diseñado supone que el entorno es **determinista**. Las soluciones a los problemas son simples secuencias de acciones, así que no pueden manejar ningún acontecimiento inesperado; además, las soluciones se ejecutan sin prestar atención a las percepciones. Los agentes que realizan sus planes con los ojos cerrados, por así decirlo, deben estar absolutamente seguros de lo que pasa (los teóricos de control lo llaman sistema de **lazo abierto**, porque ignorar las percepciones rompe el lazo entre el agente y el entorno). Todas estas suposiciones significan que tratamos con las clases más fáciles de entornos, razón por la que este capítulo aparece tan pronto en el libro. La Sección 3.6 echa una breve ojeada sobre lo que sucede cuando relajamos las suposiciones de observancia y de determinismo. Los Capítulos 12 y 17 entran más en profundidad.

LAZO ABIERTO

Problemas y soluciones bien definidos

PROBLEMA

Un **problema** puede definirse, formalmente, por cuatro componentes:

ESTADO INICIAL

- El **estado inicial** en el que comienza el agente. Por ejemplo, el estado inicial para nuestro agente en Rumanía se describe como *En*(*Arad*).

FUNCIÓN SUCESOR

- Una descripción de las posibles **acciones** disponibles por el agente. La formulación[3] más común utiliza una **función sucesor**. Dado un estado particular *x*, SUCESOR-FN(*x*) devuelve un conjunto de pares ordenados ⟨*acción, sucesor*⟩, donde cada acción es una de las acciones legales en el estado *x* y cada sucesor es un estado que puede alcanzarse desde *x,* aplicando la acción. Por ejemplo, desde el estado *En(Arad)*, la función sucesor para el problema de Rumanía devolverá

  $$\{\langle Ir(Sibiu), En(Sibiu)\rangle, \langle Ir(Timisoara), En(Timisoara)\rangle, \langle Ir(Zerind), En(Zerind)\rangle\}$$

ESPACIO DE ESTADOS

  Implícitamente el estado inicial y la función sucesor definen el **espacio de estados** del problema (el conjunto de todos los estados alcanzables desde el estado inicial). El espacio de estados forma un grafo en el cual los nodos son estados y los arcos entre los nodos son acciones. (El mapa de Rumanía que se muestra en la Figura 3.2 puede interpretarse como un grafo del espacio de estados si vemos cada carretera como dos acciones de conducir, una en cada dirección). Un **camino** en el espacio de estados es una secuencia de estados conectados por una secuencia de acciones.

CAMINO

TEST OBJETIVO

- El **test objetivo**, el cual determina si un estado es un estado objetivo. Algunas veces existe un conjunto explícito de posibles estados objetivo, y el test simplemente comprueba si el estado es uno de ellos. El objetivo del agente en Rumanía es el conjunto {*En*(*Bucarest*)}. Algunas veces el objetivo se especifica como una propiedad abstracta más que como un conjunto de estados enumerados explícitamente. Por ejemplo, en el ajedrez, el objetivo es alcanzar un estado llamado «jaque mate», donde el rey del oponente es atacado y no tiene escapatoria.

[3] Una formulación alternativa utiliza un conjunto de **operadores** que pueden aplicarse a un estado para generar así los sucesores.

Figura 3.2 Un mapa de carreteras simplificado de parte de Rumanía.

COSTO DEL CAMINO

- Una función **costo del camino** que asigna un costo numérico a cada camino. El agente resolvente de problemas elige una función costo que refleje nuestra medida de rendimiento. Para el agente que intenta llegar a Bucarest, el tiempo es esencial, así que el costo del camino puede describirse como su longitud en kilómetros. En este capítulo, suponemos que el costo del camino puede describirse como la suma de los costos de las acciones individuales a lo largo del camino. El **costo individual** de una acción a que va desde un estado x al estado y se denota por $c(x,a,y)$. Los costos individuales para Rumanía se muestran en la Figura 3.2 como las distancias de las carreteras. Suponemos que los costos son no negativos[4].

COSTO INDIVIDUAL

Los elementos anteriores definen un problema y pueden unirse en una estructura de datos simple que se dará como entrada al algoritmo resolvente del problema. Una **solución** de un problema es un camino desde el estado inicial a un estado objetivo. La calidad de la solución se mide por la función costo del camino, y una **solución óptima** tiene el costo más pequeño del camino entre todas las soluciones.

SOLUCIÓN ÓPTIMA

Formular los problemas

En la sección anterior propusimos una formulación del problema de ir a Bucarest en términos de estado inicial, función sucesor, test objetivo y costo del camino. Esta formulación parece razonable, a pesar de omitir muchos aspectos del mundo real. Para

[4] Las implicaciones de costos negativos se exploran en el Ejercicio 3.17.

comparar la descripción de un estado simple, hemos escogido, *En*(*Arad*), para un viaje real por el país, donde el estado del mundo incluye muchas cosas: los compañeros de viaje, lo que está en la radio, el paisaje, si hay algunos policías cerca, cómo está de lejos la parada siguiente, el estado de la carretera, el tiempo, etcétera. Todas estas consideraciones se dejan fuera de nuestras descripciones del estado porque son irrelevantes para el problema de encontrar una ruta a Bucarest. Al proceso de eliminar detalles de una representación se le llama **abstracción**.

ABSTRACCIÓN

Además de abstraer la descripción del estado, debemos abstraer sus acciones. Una acción de conducir tiene muchos efectos. Además de cambiar la localización del vehículo y de sus ocupantes, pasa el tiempo, consume combustible, genera contaminación, y cambia el agente (como dicen, el viaje ilustra). En nuestra formulación, tenemos en cuenta solamente el cambio en la localización. También, hay muchas acciones que omitiremos: encender la radio, mirar por la ventana, el retraso de los policías, etcétera. Y por supuesto, no especificamos acciones a nivel de «girar la rueda tres grados a la izquierda».

¿Podemos ser más precisos para definir los niveles apropiados de abstracción? Piense en los estados y las acciones abstractas que hemos elegido y que se corresponden con grandes conjuntos de estados detallados del mundo y de secuencias detalladas de acciones. Ahora considere una solución al problema abstracto: por ejemplo, la trayectoria de Arad a Sibiu a Rimnicu Vilcea a Pitesti a Bucarest. Esta solución abstracta corresponde a una gran cantidad de trayectorias más detalladas. Por ejemplo, podríamos conducir con la radio encendida entre Sibiu y Rimnicu Vilcea, y después lo apagamos para el resto del viaje. La abstracción es *válida* si podemos ampliar cualquier solución abstracta a una solución en el mundo más detallado; una condición suficiente es que para cada estado detallado de «en Arad», haya una trayectoria detallada a algún estado «en Sibiu», etcétera. La abstracción es útil si al realizar cada una de las acciones en la solución es más fácil que en el problema original; en este caso pueden ser realizadas por un agente que conduce sin búsqueda o planificación adicional. La elección de una buena abstracción implica quitar tantos detalles como sean posibles mientras que se conserve la validez y se asegure que las acciones abstractas son fáciles de realizar. Si no fuera por la capacidad de construir abstracciones útiles, los agentes inteligentes quedarían totalmente absorbidos por el mundo real.

3.2 Ejemplos de problemas

La metodología para resolver problemas se ha aplicado a un conjunto amplio de entornos. Enumeramos aquí algunos de los más conocidos, distinguiendo entre problemas de *juguete* y del *mundo-real*. Un **problema de juguete** se utiliza para ilustrar o ejercitar los métodos de resolución de problemas. Éstos se pueden describir de forma exacta y concisa. Esto significa que diferentes investigadores pueden utilizarlos fácilmente para comparar el funcionamiento de los algoritmos. Un **problema del mundo-real** es aquel en el que la gente se preocupa por sus soluciones. Ellos tienden a no tener una sola descripción, pero nosotros intentaremos dar la forma general de sus formulaciones.

PROBLEMA DE JUGUETE

PROBLEMA DEL MUNDO REAL

Problemas de juguete

El primer ejemplo que examinaremos es el **mundo de la aspiradora**, introducido en el Capítulo 2. (*Véase* Figura 2.2.) Éste puede formularse como sigue:

- **Estados:** el agente está en una de dos localizaciones, cada una de las cuales puede o no contener suciedad. Así, hay $2 \times 2^2 = 8$ posibles estados del mundo.
- **Estado inicial:** cualquier estado puede designarse como un estado inicial.
- **Función sucesor:** ésta genera los estados legales que resultan al intentar las tres acciones (*Izquierda, Derecha* y *Aspirar*). En la Figura 3.3 se muestra el espacio de estados completo.
- **Test objetivo:** comprueba si todos los cuadrados están limpios.
- **Costo del camino:** cada costo individual es 1, así que el costo del camino es el número de pasos que lo compone.

Comparado con el mundo real, este problema de juguete tiene localizaciones discretas, suciedad discreta, limpieza fiable, y nunca se ensucia una vez que se ha limpiado. (En la Sección 3.6 relajaremos estas suposiciones). Una cosa a tener en cuenta es que el estado está determinado por la localización del agente y por las localizaciones de la suciedad. Un entorno grande con n localizaciones tiene $n\ 2^n$ estados.

Figura 3.3 Espacio de estados para el mundo de la aspiradora. Los arcos denotan las acciones: I = *Izquierda*, D = *Derecha*, A = *Aspirar*.

8-PUZZLE

El **8-puzle**, la Figura 3.4 muestra un ejemplo, consiste en un tablero de 3×3 con ocho fichas numeradas y un espacio en blanco. Una ficha adyacente al espacio en blanco puede deslizarse a éste. La meta es alcanzar el estado objetivo especificado, tal como se muestra a la derecha de la figura. La formulación estándar es como sigue:

- **Estados:** la descripción de un estado especifica la localización de cada una de las ocho fichas y el blanco en cada uno de los nueve cuadrados.

- **Estado inicial:** cualquier estado puede ser un estado inicial. Nótese que cualquier objetivo puede alcanzarse desde exactamente la mitad de los estados iniciales posibles (Ejercicio 3.4).
- **Función sucesor:** ésta genera los estados legales que resultan de aplicar las cuatro acciones (mover el blanco a la *Izquierda*, *Derecha*, *Arriba* y *Abajo*).
- **Test objetivo:** comprueba si el estado coincide con la configuración objetivo que se muestra en la Figura 3.4. (son posibles otras configuraciones objetivo).
- **Costo del camino:** el costo de cada paso del camino tiene valor 1, así que el costo del camino es el número de pasos.

¿Qué abstracciones se han incluido? Las acciones se han abstraído a los estados iniciales y finales, ignorando las localizaciones intermedias en donde se deslizan los bloques. Hemos abstraído acciones como la de sacudir el tablero cuando las piezas no se pueden mover, o extraer las piezas con un cuchillo y volverlas a poner. Nos dejan con una descripción de las reglas del puzle que evitan todos los detalles de manipulaciones físicas.

Figura 3.4 Un ejemplo típico del 8-puzle.

PIEZAS DESLIZANTES

El 8-puzle pertenece a la familia de puzles con **piezas deslizantes**, los cuales a menudo se usan como problemas test para los nuevos algoritmos de IA. Esta clase general se conoce por ser NP completa, así que no esperamos encontrar métodos perceptiblemente mejores (en el caso peor) que los algoritmos de búsqueda descritos en este capítulo y en el siguiente. El 8-puzle tiene $9!/2 = 181{,}440$ estados alcanzables y se resuelve fácilmente. El 15 puzle (sobre un tablero de 4×4) tiene alrededor de 1,3 trillones de estados, y configuraciones aleatorias pueden resolverse óptimamente en pocos milisegundos por los mejores algoritmos de búsqueda. El 24 puzle (sobre un tablero de 5×5) tiene alrededor de 10^{25} estados, y configuraciones aleatorias siguen siendo absolutamente difíciles de resolver de manera óptima con los computadores y algoritmos actuales.

PROBLEMA 8-REINAS

El objetivo del **problema de las 8-reinas** es colocar las ocho reinas en un tablero de ajedrez de manera que cada reina no ataque a ninguna otra. (Una reina ataca alguna pieza si está en la misma fila, columna o diagonal.) La Figura 3.5 muestra una configuración que no es solución: la reina en la columna de más a la derecha está atacando a la reina de arriba a la izquierda.

Figura 3.5 Casi una solución del problema de las 8-reinas. (La solución se deja como ejercicio.)

Aunque existen algoritmos eficientes específicos para este problema y para el problema general de las n reinas, sigue siendo un problema test interesante para los algoritmos de búsqueda. Existen dos principales formulaciones. Una **formulación incremental** que implica a operadores que aumentan la descripción del estado, comenzando con un estado vacío; para el problema de las 8-reinas, esto significa que cada acción añade una reina al estado. Una **formulación completa de estados** comienza con las ocho reinas en el tablero y las mueve. En cualquier caso, el coste del camino no tiene ningún interés porque solamente cuenta el estado final. La primera formulación incremental que se puede intentar es la siguiente:

FORMULACIÓN INCREMENTAL

FORMULACIÓN COMPLETA DE ESTADOS

- **Estados:** cualquier combinación de cero a ocho reinas en el tablero es un estado.
- **Estado inicial:** ninguna reina sobre el tablero.
- **Función sucesor:** añadir una reina a cualquier cuadrado vacío.
- **Test objetivo:** ocho reinas sobre el tablero, ninguna es atacada.

En esta formulación, tenemos $64 \cdot 63 \cdots 57 \approx 3 \times 10^{14}$ posibles combinaciones a investigar. Una mejor formulación deberá prohibir colocar una reina en cualquier cuadrado que esté realmente atacado:

- **Estados:** son estados, la combinación de n reinas ($0 \leq n \leq 8$), una por columna desde la columna más a la izquierda, sin que una reina ataque a otra.
- **Función sucesor:** añadir una reina en cualquier cuadrado en la columna más a la izquierda vacía tal que no sea atacada por cualquier otra reina.

Esta formulación reduce el espacio de estados de las 8-reinas de 3×10^{14} a 2.057, y las soluciones son fáciles de encontrar. Por otra parte, para 100 reinas la formulación inicial tiene 10^{400} estados mientras que las formulaciones mejoradas tienen cerca de 10^{52} estados (Ejercicio 3.5). Ésta es una reducción enorme, pero el espacio de estados mejorado sigue siendo demasiado grande para los algoritmos de este capítulo. El Capítulo 4

describe la formulación completa de estados y el Capítulo 5 nos da un algoritmo sencillo que hace el problema de un millón de reinas fácil de resolver.

Problemas del mundo real

PROBLEMA DE BÚSQUEDA DE UNA RUTA

Hemos visto cómo el **problema de búsqueda de una ruta** está definido en términos de posiciones y transiciones a lo largo de ellas. Los algoritmos de búsqueda de rutas se han utilizado en una variedad de aplicaciones, tales como rutas en redes de computadores, planificación de operaciones militares, y en sistemas de planificación de viajes de líneas aéreas. Estos problemas son complejos de especificar. Consideremos un ejemplo simplificado de un problema de viajes de líneas aéreas que especificamos como:

- **Estados:** cada estado está representado por una localización (por ejemplo, un aeropuerto) y la hora actual.
- **Estado inicial:** especificado por el problema.
- **Función sucesor:** devuelve los estados que resultan de tomar cualquier vuelo programado (quizá más especificado por la clase de asiento y su posición) desde el aeropuerto actual a otro, que salgan a la hora actual más el tiempo de tránsito del aeropuerto.
- **Test objetivo:** ¿tenemos nuestro destino para una cierta hora especificada?
- **Costo del camino:** esto depende del costo en dinero, tiempo de espera, tiempo del vuelo, costumbres y procedimientos de la inmigración, calidad del asiento, hora, tipo de avión, kilometraje del aviador experto, etcétera.

Los sistemas comerciales de viajes utilizan una formulación del problema de este tipo, con muchas complicaciones adicionales para manejar las estructuras bizantinas del precio que imponen las líneas aéreas. Cualquier viajero experto sabe, sin embargo, que no todo el transporte aéreo va según lo planificado. Un sistema realmente bueno debe incluir planes de contingencia (tales como reserva en vuelos alternativos) hasta el punto de que éstos estén justificados por el coste y la probabilidad de la falta del plan original.

PROBLEMAS TURÍSTICOS

Los **problemas turísticos** están estrechamente relacionados con los problemas de búsqueda de una ruta, pero con una importante diferencia. Consideremos, por ejemplo, el problema, «visitar cada ciudad de la Figura 3.2 al menos una vez, comenzando y finalizando en Bucarest». Como en la búsqueda de rutas, las acciones corresponden a viajes entre ciudades adyacentes. El espacio de estados, sin embargo, es absolutamente diferente. Cada estado debe incluir no sólo la localización actual sino también *las ciudades que el agente ha visitado*. El estado inicial sería «En Bucarest; visitado {Bucarest}», un estado intermedio típico sería «En Vaslui; visitado {Bucarest, Urziceni, Vaslui}», y el test objetivo comprobaría si el agente está en Bucarest y ha visitado las 20 ciudades.

PROBLEMA DEL VIAJANTE DE COMERCIO

El **problema del viajante de comercio** (PVC) es un problema de ruta en la que cada ciudad es visitada exactamente una vez. La tarea principal es encontrar el viaje *más corto*. El problema es de tipo NP duro, pero se ha hecho un gran esfuerzo para mejorar las capacidades de los algoritmos del PVC. Además de planificación de los viajes del viajante de comercio, estos algoritmos se han utilizado para tareas tales como la plani-

ficación de los movimientos de los taladros de un circuito impreso y para abastecer de máquinas a las tiendas.

DISTRIBUCIÓN VLSI

Un problema de **distribución VLSI** requiere la colocación de millones de componentes y de conexiones en un chip verificando que el área es mínima, que se reduce al mínimo el circuito, que se reduce al mínimo las capacitaciones, y se maximiza la producción de fabricación. El problema de la distribución viene después de la fase de diseño lógico, y está dividido generalmente en dos partes: **distribución de las celdas** y **dirección del canal**. En la distribución de las celdas, los componentes primitivos del circuito se agrupan en las celdas, cada una de las cuales realiza una cierta función. Cada celda tiene una característica fija (el tamaño y la forma) y requiere un cierto número de conexiones a cada una de las otras celdas. El objetivo principal es colocar las celdas en el chip de manera que no se superpongan y que quede espacio para que los alambres que conectan celdas puedan colocarse entre ellas. La dirección del canal encuentra una ruta específica para cada alambre por los espacios entre las celdas. Estos problemas de búsqueda son extremadamente complejos, pero definitivamente dignos de resolver. En el Capítulo 4, veremos algunos algoritmos capaces de resolverlos.

NAVEGACIÓN DE UN ROBOT

La **navegación de un robot** es una generalización del problema de encontrar una ruta descrito anteriormente. Más que un conjunto discreto de rutas, un robot puede moverse en un espacio continuo con (en principio) un conjunto infinito de acciones y estados posibles. Para un robot circular que se mueve en una superficie plana, el espacio es esencialmente de dos dimensiones. Cuando el robot tiene manos y piernas o ruedas que se deben controlar también, el espacio de búsqueda llega a ser de muchas dimensiones. Lo que se requiere es que las técnicas avanzadas hagan el espacio de búsqueda finito. Examinaremos algunos de estos métodos en el Capítulo 25. Además de la complejidad del problema, los robots reales también deben tratar con errores en las lecturas de los sensores y controles del motor.

SECUENCIACIÓN PARA EL ENSAMBLAJE AUTOMÁTICO

La **secuenciación para el ensamblaje automático** por un robot de objetos complejos fue demostrado inicialmente por FREDDY (Michie, 1972). Los progresos desde entonces han sido lentos pero seguros, hasta el punto de que el ensamblaje de objetos tales como motores eléctricos son económicamente factibles. En los problemas de ensamblaje, lo principal es encontrar un orden en los objetos a ensamblar. Si se elige un orden equivocado, no habrá forma de añadir posteriormente una parte de la secuencia sin deshacer el trabajo ya hecho. Verificar un paso para la viabilidad de la sucesión es un problema de búsqueda geométrico difícil muy relacionado con la navegación del robot. Así, la generación de sucesores legales es la parte costosa de la secuenciación para el ensamblaje. Cualquier algoritmo práctico debe evitar explorar todo, excepto una fracción pequeña del espacio de estados. Otro problema de ensamblaje importante es el **diseño de proteínas**, en el que el objetivo es encontrar una secuencia de aminoácidos que se plegarán en una proteína de tres dimensiones con las propiedades adecuadas para curar alguna enfermedad.

DISEÑO DE PROTEÍNAS

BÚSQUEDA EN INTERNET

En la actualidad, se ha incrementado la demanda de robots *software* que realicen la **búsqueda en Internet**, la búsqueda de respuestas a preguntas, de información relacionada o para compras. Esto es una buena aplicación para las técnicas de búsqueda, porque es fácil concebir Internet como un grafo de nodos (páginas) conectadas por arcos. Una descripción completa de búsqueda en Internet se realiza en el Capítulo 10.

3.3 Búsqueda de soluciones

ÁRBOL DE BÚSQUEDA

Hemos formulado algunos problemas, ahora necesitamos resolverlos. Esto se hace mediante búsqueda a través del espacio de estados. Este capítulo se ocupa de las técnicas de búsqueda que utilizan un **árbol de búsqueda** explícito generado por el estado inicial y la función sucesor, definiendo así el espacio de estados. En general, podemos tener un grafo de búsqueda más que un árbol, cuando el mismo estado puede alcanzarse desde varios caminos. Aplazamos, hasta la Sección 3.5, el tratar estas complicaciones importantes.

NODO DE BÚSQUEDA

EXPANDIR

GENERAR

La Figura 3.6 muestra algunas de las expansiones en el árbol de búsqueda para encontrar una camino desde Arad a Bucarest. La raíz del árbol de búsqueda es el **nodo de búsqueda** que corresponde al estado inicial, *En*(*Arad*). El primer paso es comprobar si éste es un estado objetivo. Claramente es que no, pero es importante comprobarlo de modo que podamos resolver problemas como «comenzar en Arad, consigue Arad». Como no estamos en un estado objetivo, tenemos que considerar otros estados. Esto se hace **expandiendo** el estado actual; es decir aplicando la función sucesor al estado actual y **generar** así un nuevo conjunto de estados. En este caso, conseguimos tres nuevos estados: *En*(*Sibiu*), *En*(*Timisoara*) y *En*(*Zerind*). Ahora debemos escoger cuál de estas tres posibilidades podemos considerar.

ESTRATEGIA DE BÚSQUEDA

Esto es la esencia de la búsqueda, llevamos a cabo una opción y dejamos de lado las demás para más tarde, en caso de que la primera opción no conduzca a una solución. Supongamos que primero elegimos Sibiu. Comprobamos si es un estado objetivo (que no lo es) y entonces expandimos para conseguir *En*(*Arad*), *En*(*Fagaras*), *En*(*Oradea*) y *En*(*RimnicuVilcea*). Entonces podemos escoger cualquiera de estas cuatro o volver atrás y escoger Timisoara o Zerind. Continuamos escogiendo, comprobando y expandiendo hasta que se encuentra una solución o no existen más estados para expandir. El estado a expandir está determinado por la **estrategia de búsqueda**. La Figura 3.7 describe informalmente el algoritmo general de búsqueda en árboles.

Es importante distinguir entre el espacio de estados y el árbol de búsqueda. Para el problema de búsqueda de un ruta, hay solamente 20 estados en el espacio de estados, uno por cada ciudad. Pero hay un número infinito de caminos en este espacio de estados, así que el árbol de búsqueda tiene un número infinito de nodos. Por ejemplo, los tres caminos Arad-Sibiu, Arad-Sibiu-Arad, Arad-Sibiu-Arad-Sibiu son los tres primeros caminos de una secuencia infinita de caminos. (Obviamente, un buen algoritmo de búsqueda evita seguir tales trayectorias; La Sección 3.5 nos muestra cómo hacerlo).

Hay muchas formas de representar los nodos, pero vamos a suponer que un nodo es una estructura de datos con cinco componentes:

— ESTADO: el estado, del espacio de estados, que corresponde con el nodo;
— NODO PADRE: el nodo en el árbol de búsqueda que ha generado este nodo;
— ACCIÓN: la acción que se aplicará al padre para generar el nodo;
— COSTO DEL CAMINO: el costo, tradicionalmente denotado por $g(n)$, de un camino desde el estado inicial al nodo, indicado por los punteros a los padres; y
— PROFUNDIDAD: el número de pasos a los largo del camino desde el estado inicial.

Es importante recordar la distinción entre nodos y estados. Un nodo es una estructura de datos usada para representar el árbol de búsqueda. Un estado corresponde a una

Figura 3.6 Árboles de búsqueda parciales para encontrar una ruta desde Arad hasta Bucarest. Se sombrean los nodos que se han expandido; los nodos que se han generado pero todavía no se han expandido se rodean en negrita; los nodos que todavía no se han generado se marcan con líneas discontinuas.

función BÚSQUEDA-ÁRBOLES(*problema,estrategia*) **devuelve** una solución o fallo
  inicializa el árbol de búsqueda usando el estado inicial del *problema*
  bucle hacer
    si no hay candidatos para expandir **entonces devolver** fallo
    escoger, de acuerdo a la *estrategia*, un nodo hoja para expandir
    si el nodo contiene un estado objetivo **entonces devolver** la correspondiente solución
    en otro caso expandir el nodo y añadir los nodos resultado al árbol de búsqueda

Figura 3.7 Descripción informal del algoritmo general de búsqueda en árboles.

configuración del mundo. Así, los nodos están en caminos particulares, según lo definido por los punteros del nodo padre, mientras que los estados no lo están. En la Figura 3.8 se representa la estructura de datos del nodo.

FRONTERA

También necesitamos representar la colección de nodos que se han generado pero todavía no se han expandido – a esta colección se le llama **frontera**. Cada elemento de

Figura 3.8 Los nodos son estructuras de datos a partir de los cuales se construye el árbol de búsqueda. Cada uno tiene un padre, un estado y varios campos. Las flechas señalan del hijo al padre.

NODO HOJA

la frontera es un **nodo hoja**, es decir, un nodo sin sucesores en el árbol. En la Figura 3.6, la frontera de cada árbol consiste en los nodos dibujados con líneas discontinuas. La representación más simple de la frontera sería como un conjunto de nodos. La estrategia de búsqueda será una función que seleccione de este conjunto el siguiente nodo a expandir. Aunque esto sea conceptualmente sencillo, podría ser computacionalmente costoso, porque la función estrategia quizá tenga que mirar cada elemento del conjunto para escoger el mejor. Por lo tanto, nosotros asumiremos que la colección de nodos se implementa como una **cola**. Las operaciones en una cola son como siguen:

COLA

— HACER-COLA(*elemento*, …) crea una cola con el(los) elemento(s) dado(s).
— VACIA?(*cola*) devuelve verdadero si no hay ningún elemento en la cola.
— PRIMERO(*cola*) devuelve el primer elemento de la cola.
— BORRAR-PRIMERO(*cola*) devuelve PRIMERO(*cola*) y lo borra de la cola.
— INSERTA(*elemento,cola*) inserta un elemento en la cola y devuelve la cola resultado. (Veremos que tipos diferentes de colas insertan los elementos en órdenes diferentes.)
— INSERTAR-TODO(*elementos,cola*) inserta un conjunto de elementos en la cola y devuelve la cola resultado.

Con estas definiciones, podemos escribir una versión más formal del algoritmo general de búsqueda en árboles. Se muestra en la Figura 3.9.

Medir el rendimiento de la resolución del problema

La salida del algoritmo de resolución de problemas es *fallo* o una solución. (Algunos algoritmos podrían caer en un bucle infinito y nunca devolver una salida.) Evaluaremos el rendimiento de un algoritmo de cuatro formas:

COMPLETITUD

OPTIMIZACIÓN

- **Completitud:** ¿está garantizado que el algoritmo encuentre una solución cuando esta exista?
- **Optimización:** ¿encuentra la estrategia la solución óptima, según lo definido en la página 62?

```
función BÚSQUEDA-ÁRBOLES(problema,frontera) devuelve una solución o fallo
  frontera ← INSERTA(HACER-NODO(ESTADO-INICIAL[problema]), frontera)
  hacer bucle
    si VACIA?(frontera) entonces devolver fallo.
    nodo ← BORRAR-PRIMERO(frontera)
    si TEST-OBJETIVO[problema] aplicado al ESTADO[nodo] es cierto
      entonces devolver SOLUCIÓN(nodo)
    frontera ← INSERTAR-TODO(EXPANDIR(nodo,problema),frontera)
```

```
función EXPANDIR(nodo,problema) devuelve un conjunto de nodos
  sucesores ← conjunto vacío
  para cada (acción,resultado) en SUCESOR-FN[problema](ESTADO[nodo]) hacer
    s ← un nuevo NODO
    ESTADO[s] ← resultado
    NODO-PADRE[s] ← nodo
    ACCIÓN[s] ← acción
    COSTO-CAMINO[s] ← COSTO-CAMINO[nodo] + COSTO-INDIVIDUAL(nodo,acción,s)
    PROFUNDIDAD[s] ← PROFUNDIDAD[nodo] + 1
    añadir s a sucesores
  devolver sucesores
```

Figura 3.9 Algoritmo general de búsqueda en árboles. (Notemos que el argumento *frontera* puede ser una cola vacía, y el tipo de cola afectará al orden de la búsqueda.) La función SOLUCIÓN devuelve la secuencia de acciones obtenida de la forma punteros al padre hasta la raíz.

COMPLEJIDAD EN TIEMPO

COMPLEJIDAD EN ESPACIO

- **Complejidad en tiempo:** ¿cuánto tarda en encontrar una solución?
- **Complejidad en espacio:** ¿cuánta memoria se necesita para el funcionamiento de la búsqueda?

La complejidad en tiempo y espacio siempre se considera con respecto a alguna medida de la dificultad del problema. En informática teórica, la medida es el tamaño del grafo del espacio de estados, porque el grafo se ve como una estructura de datos explícita que se introduce al programa de búsqueda. (El mapa de Rumanía es un ejemplo de esto.) En IA, donde el grafo está representado de forma implícita por el estado inicial y la función sucesor y frecuentemente es infinito, la complejidad se expresa en términos de tres cantidades: b, el **factor de ramificación** o el máximo número de sucesores de cualquier nodo; d, la profundidad del nodo objetivo más superficial; y m, la longitud máxima de cualquier camino en el espacio de estados.

FACTOR DE RAMIFICACIÓN

El tiempo a menudo se mide en términos de número de nodos generados[5] durante la búsqueda, y el espacio en términos de máximo número de nodos que se almacena en memoria.

COSTO DE LA BÚSQUEDA

Para valorar la eficacia de un algoritmo de búsqueda, podemos considerar el **costo de la búsqueda** (que depende típicamente de la complejidad en tiempo pero puede in-

[5] Algunos textos miden el tiempo en términos del número de las expansiones del nodo. Las dos medidas se diferencian como mucho en un factor b. A nosotros nos parece que el tiempo de ejecución de la expansión del nodo aumenta con el número de nodos generados en esa expansión.

COSTE TOTAL

cluir también un término para el uso de la memoria) o podemos utilizar el **coste total**, que combina el costo de la búsqueda y el costo del camino solución encontrado. Para el problema de encontrar una ruta desde Arad hasta Bucarest, el costo de la búsqueda es la cantidad de tiempo que ha necesitado la búsqueda y el costo de la solución es la longitud total en kilómetros del camino. Así, para el cálculo del coste total, tenemos que sumar kilómetros y milisegundos. No hay ninguna conversión entre los dos, pero quizá sea razonable, en este caso, convertir kilómetros en milisegundos utilizando una estimación de la velocidad media de un coche (debido a que el tiempo es lo que cuida el agente.) Esto permite al agente encontrar un punto óptimo de intercambio en el cual el cálculo adicional para encontrar que un camino más corto llegue a ser contraproducente. El problema más general de intercambios entre bienes diferentes será tratado en el Capítulo 16.

3.4 Estrategias de búsqueda no informada

BÚSQUEDA NO INFORMADA

BÚSQUEDA INFORMADA

BÚSQUEDA HEURÍSTICA

Esta sección trata cinco estrategias de búsqueda englobadas bajo el nombre de **búsqueda no informada** (llamada también **búsqueda a ciegas**). El término significa que ellas no tienen información adicional acerca de los estados más allá de la que proporciona la definición del problema. Todo lo que ellas pueden hacer es generar los sucesores y distinguir entre un estado objetivo de uno que no lo es. Las estrategias que saben si un estado no objetivo es «más prometedor» que otro se llaman **búsqueda informada** o **búsqueda heurística**; éstas serán tratadas en el Capítulo 4. Todas las estrategias se distinguen por el *orden* de expansión de los nodos.

Búsqueda primero en anchura

BÚSQUEDA PRIMERO EN ANCHURA

La **búsqueda primero en anchura** es una estrategia sencilla en la que se expande primero el nodo raíz, a continuación se expanden todos los sucesores del nodo raíz, después *sus* sucesores, etc. En general, se expanden todos los nodos a una profundidad en el árbol de búsqueda antes de expandir cualquier nodo del próximo nivel.

La búsqueda primero en anchura se puede implementar llamando a la BÚSQUEDA-ÁRBOLES con una frontera vacía que sea una cola primero en entrar primero en salir (FIFO), asegurando que los nodos primeros visitados serán los primeros expandidos. En otras palabras, llamando a la BÚSQUEDA-ÁRBOLES(*problema*,COLA-FIFO()) resulta una búsqueda primero en anchura. La cola FIFO pone todos los nuevos sucesores generados al final de la cola, lo que significa que los nodos más superficiales se expanden antes que los nodos más profundos. La Figura 3.10 muestra el progreso de la búsqueda en un árbol binario sencillo.

Evaluemos la búsqueda primero en anchura usando los cuatro criterios de la sección anterior. Podemos ver fácilmente que es *completa* (si el nodo objetivo más superficial está en una cierta profundidad finita d, la búsqueda primero en anchura lo encontrará después de expandir todos los nodos más superficiales, con tal que el factor de ramificación b sea finito). El nodo objetivo más superficial no es necesariamente el óptimo;

Figura 3.10 Búsqueda primero en anchura sobre un árbol binario sencillo. En cada etapa, el próximo nodo a expandir se indica con una marca.

técnicamente, la búsqueda primero en anchura es óptima si el costo del camino es una función no decreciente de la profundidad del nodo (por ejemplo, cuando todas las acciones tienen el mismo coste).

Hasta ahora, la información sobre la búsqueda primero en anchura ha sido buena. Para ver por qué no es siempre la estrategia a elegir, tenemos que considerar la cantidad de tiempo y memoria que utiliza para completar una búsqueda. Para hacer esto, consideramos un espacio de estados hipotético donde cada estado tiene b sucesores. La raíz del árbol de búsqueda genera b nodos en el primer nivel, cada uno de ellos genera b nodos más, teniendo un total de b^2 en el segundo nivel. Cada uno de estos genera b nodos más, teniendo b^3 nodos en el tercer nivel, etcétera. Ahora supongamos que la solución está a una profundidad d. En el peor caso, expandiremos todos excepto el último nodo en el nivel d (ya que el objetivo no se expande), generando $b^{d+1} - b$ nodos en el nivel $d + 1$. Entonces el número total de nodos generados es

$$b + b^2 + b^3 + ... + b^d + (b^{d+1} - b) = O(b^{d+1}).$$

Cada nodo generado debe permanecer en la memoria, porque o es parte de la frontera o es un antepasado de un nodo de la frontera. La complejidad en espacio es, por lo tanto, la misma que la complejidad en tiempo (más un nodo para la raíz).

Los que hacen análisis de complejidad están preocupados (o emocionados, si les gusta el desafío) por las cotas de complejidad exponencial como $O(b^{d+1})$. La Figura 3.11 muestra por qué. Se enumera el tiempo y la memoria requerida para una búsqueda primero en anchura con el factor de ramificación $b = 10$, para varios valores de profundi-

Profundidad	Nodos	Tiempo	Memoria
2	1.100	11 segundos	1 megabyte
4	111.100	11 segundos	106 megabytes
6	10^7	19 minutos	10 gigabytes
8	10^9	31 horas	1 terabytes
10	10^{11}	129 días	101 terabytes
12	10^{13}	35 años	10 petabytes
14	10^{15}	3.523 años	1 exabyte

Figura 3.11 Requisitos de tiempo y espacio para la búsqueda primero en anchura. Los números que se muestran suponen un factor de ramificación $b = 10$; 10.000 nodos/segundo; 1.000 bytes/nodo.

dad d de la solución. La tabla supone que se pueden generar 10.000 nodos por segundo y que un nodo requiere 1.000 bytes para almacenarlo. Muchos problemas de búsqueda quedan aproximadamente dentro de estas suposiciones (más o menos un factor de 100) cuando se ejecuta en un computador personal moderno.

Hay dos lecciones que debemos aprender de la Figura 3.11. Primero, *son un problema más grande los requisitos de memoria para la búsqueda primero en anchura que el tiempo de ejecución*. 31 horas no sería demasiado esperar para la solución de un problema importante a profundidad ocho, pero pocos computadores tienen suficientes terabytes de memoria principal que lo admitieran. Afortunadamente, hay otras estrategias de búsqueda que requieren menos memoria.

La segunda lección es que los requisitos de tiempo son todavía un factor importante. Si su problema tiene una solución a profundidad 12, entonces (dadas nuestras suposiciones) llevará 35 años encontrarla por la búsqueda primero en anchura (o realmente alguna búsqueda sin información). En general, *los problemas de búsqueda de complejidad-exponencial no pueden resolverse por métodos sin información, salvo casos pequeños*.

Búsqueda de costo uniforme

La búsqueda primero en anchura es óptima cuando todos los costos son iguales, porque siempre expande el nodo no expandido más superficial. Con una extensión sencilla, podemos encontrar un algoritmo que es óptimo con cualquier función costo. En vez de expandir el nodo más superficial, la **búsqueda de costo uniforme** expande el nodo n con el camino de costo más pequeño. Notemos que si todos los costos son iguales, es idéntico a la búsqueda primero en anchura.

BÚSQUEDA DE COSTO UNIFORME

La búsqueda de costo uniforme no se preocupa por el número de pasos que tiene un camino, pero sí sobre su coste total. Por lo tanto, éste se meterá en un bucle infinito si expande un nodo que tiene una acción de coste cero que conduzca de nuevo al mismo estado (por ejemplo, una acción *NoOp*). Podemos garantizar completitud si el costo de cada paso es mayor o igual a alguna constante positiva pequeña ϵ. Esta condición es también suficiente para asegurar optimización. Significa que el costo de un camino siempre aumenta cuando vamos por él. De esta propiedad, es fácil ver que el algoritmo expande nodos que incrementan el coste del camino. Por lo tanto, el primer nodo objetivo seleccionado para la expansión es la solución óptima. (Recuerde que la búsqueda en árboles aplica el test objetivo sólo a los nodos que son seleccionados para la expansión.) Recomendamos probar el algoritmo para encontrar el camino más corto a Bucarest.

La búsqueda de costo uniforme está dirigida por los costos de los caminos más que por las profundidades, entonces su complejidad no puede ser fácilmente caracterizada en términos de b y d. En su lugar, C^* es el costo de la solución óptima, y se supone que cada acción cuesta al menos ϵ. Entonces la complejidad en tiempo y espacio del peor caso del algoritmo es $O(b^{\lceil C^*/\epsilon \rceil})$, la cual puede ser mucho mayor que b^d. Esto es porque la búsqueda de costo uniforme, y a menudo lo hace, explora los árboles grandes en pequeños pasos antes de explorar caminos que implican pasos grandes y quizás útiles. Cuando todos los costos son iguales, desde luego, la $b^{\lceil C^*/\epsilon \rceil}$ es justamente b^d.

Búsqueda primero en profundidad

BÚSQUEDA PRIMERO EN PROFUNDIDAD

La **búsqueda primero en profundidad** siempre expande el nodo *más profundo* en la frontera actual del árbol de búsqueda. El progreso de la búsqueda se ilustra en la Figura 3.12. La búsqueda procede inmediatamente al nivel más profundo del árbol de búsqueda, donde los nodos no tienen ningún sucesor. Cuando esos nodos se expanden, son quitados de la frontera, así entonces la búsqueda «retrocede» al siguiente nodo más superficial que todavía tenga sucesores inexplorados.

Esta estrategia puede implementarse por la BÚSQUEDA-ÁRBOLES con una cola último en entrar primero en salir (LIFO), también conocida como una pila. Como una alternativa a la implementación de la BÚSQUEDA-ÁRBOLES, es común aplicar la búsqueda primero en profundidad con una función recursiva que se llama en cada uno de sus hijos. (Un algoritmo primero en profundidad recursivo incorporando un límite de profundidad se muestra en la Figura 3.13.)

La búsqueda primero en profundidad tiene unos requisitos muy modestos de memoria. Necesita almacenar sólo un camino desde la raíz a un nodo hoja, junto con los nodos hermanos restantes no expandidos para cada nodo del camino. Una vez que un nodo se

Figura 3.12 Búsqueda primero en profundidad sobre un árbol binario. Los nodos que se han expandido y no tienen descendientes en la frontera se pueden quitar de la memoria; estos se muestran en negro. Los nodos a profundidad 3 se suponen que no tienen sucesores y *M* es el nodo objetivo.

```
función BÚSQUEDA-PROFUNDIDAD-LIMITADA(problema,límite)    devuelve una solución, o
  fallo/corte
  devolver BPL-RECURSIVO(HACER-NODO(ESTADO-INICIAL[problema]),problema,límite)

función BPL-RECURSIVO(nodo,problema,límite) devuelve una solución, o fallo/corte
  ocurrió un corte ← falso
  si TEST-OBJETIVO[problema](ESTADO[nodo]) entonces devolver SOLUCIÓN(nodo)
  en caso contrario si PROFUNDIDAD[nodo] = límite entonces devolver corte
  en caso contrario para cada sucesor en EXPANDIR(nodo,problema) hacer
    resultado ← BPL-RECURSIVO(sucesor,problema,límite)
    si resultado = corte entonces ocurrió un corte ← verdadero
    en otro caso si resultado ≠ fallo entonces devolver resultado
  si ocurrió un corte? entonces devolver corte en caso contrario devolver fallo
```

Figura 3.13 Implementación recursiva de la búsqueda primero en profundidad.

ha expandido, se puede quitar de la memoria tan pronto como todos su descendientes han sido explorados. (*Véase* Figura 3.12.) Para un espacio de estados con factor de ramificación b y máxima profundidad m, la búsqueda primero en profundidad requiere almacenar sólo $bm + 1$ nodos. Utilizando las mismas suposiciones que con la Figura 3.11, y suponiendo que los nodos a la misma profundidad que el nodo objetivo no tienen ningún sucesor, nos encontramos que la búsqueda primero en profundidad requeriría 118 kilobytes en vez de diez petabytes a profundidad $d = 12$, un factor de diez billones de veces menos de espacio.

BÚSQUEDA HACIA ATRÁS

Una variante de la búsqueda primero en profundidad, llamada **búsqueda hacia atrás,** utiliza todavía menos memoria. En la búsqueda hacia atrás, sólo se genera un sucesor a la vez; cada nodo parcialmente expandido recuerda qué sucesor se expande a continuación. De esta manera, sólo se necesita $O(m)$ memoria más que el $O(bm)$ anterior. La búsqueda hacia atrás facilita aún otro ahorro de memoria (y ahorro de tiempo): la idea de generar un sucesor modificando directamente la descripción actual del estado más que copiarlo. Esto reduce los requerimientos de memoria a solamente una descripción del estado y $O(m)$ acciones. Para hacer esto, debemos ser capaces de deshacer cada modificación cuando volvemos hacia atrás para generar el siguiente sucesor. Para problemas con grandes descripciones de estados, como el ensamblaje en robótica, estas técnicas son críticas para tener éxito.

El inconveniente de la búsqueda primero en profundidad es que puede hacer una elección equivocada y obtener un camino muy largo (o infinito) aun cuando una elección diferente llevaría a una solución cerca de la raíz del árbol de búsqueda. Por ejemplo, en la Figura 3.12, la búsqueda primero en profundidad explorará el subárbol izquierdo entero incluso si el nodo C es un nodo objetivo. Si el nodo J fuera también un nodo objetivo, entonces la búsqueda primero en profundidad lo devolvería como una solución; de ahí, que la búsqueda primero en profundidad no es óptima. Si el subárbol izquierdo fuera de profundidad ilimitada y no contuviera ninguna solución, la búsqueda primero en profundidad nunca terminaría; de ahí, que no es completo. En el caso peor, la búsqueda primero en profundidad generará todos los nodos $O(b^m)$ del árbol de búsqueda, don-

de m es la profundidad máxima de cualquier nodo. Nótese que m puede ser mucho más grande que d (la profundidad de la solución más superficial), y es infinito si el árbol es ilimitado.

Búsqueda de profundidad limitada

BÚSQUEDA DE PROFUNDIDAD LIMITADA

Se puede aliviar el problema de árboles ilimitados aplicando la búsqueda primero en profundidad con un límite de profundidad ℓ predeterminado. Es decir, los nodos a profundidad ℓ se tratan como si no tuvieran ningún sucesor. A esta aproximación se le llama **búsqueda de profundidad limitada**. El límite de profundidad resuelve el problema del camino infinito. Lamentablemente, también introduce una fuente adicional de incompletitud si escogemos $\ell < d$, es decir, el objetivo está fuera del límite de profundidad. (Esto no es improbable cuando d es desconocido.) La búsqueda de profundidad limitada también será no óptima si escogemos $\ell > d$. Su complejidad en tiempo es $O(b^\ell)$ y su complejidad en espacio es $O(b\ell)$. La búsqueda primero en profundidad puede verse como un caso especial de búsqueda de profundidad limitada con $\ell = \infty$.

DIÁMETRO

A veces, los límites de profundidad pueden estar basados en el conocimiento del problema. Por ejemplo, en el mapa de Rumanía hay 20 ciudades. Por lo tanto, sabemos que si hay una solución, debe ser de longitud 19 como mucho, entonces $\ell = 19$ es una opción posible. Pero de hecho si estudiáramos el mapa con cuidado, descubriríamos que cualquier ciudad puede alcanzarse desde otra como mucho en nueve pasos. Este número, conocido como el **diámetro** del espacio de estados, nos da un mejor límite de profundidad, que conduce a una búsqueda con profundidad limitada más eficiente. Para la mayor parte de problemas, sin embargo, no conoceremos un límite de profundidad bueno hasta que hayamos resuelto el problema.

La búsqueda de profundidad limitada puede implementarse con una simple modificación del algoritmo general de búsqueda en árboles o del algoritmo recursivo de búsqueda primero en profundidad. En la Figura 3.13 se muestra el pseudocódigo de la búsqueda recursiva de profundidad limitada. Notemos que la búsqueda de profundidad limitada puede terminar con dos clases de fracaso: el valor de *fracaso* estándar indicando que no hay ninguna solución; el valor de *corte* indicando que no hay solución dentro del límite de profundidad.

Búsqueda primero en profundidad con profundidad iterativa

BÚSQUEDA CON PROFUNDIDAD ITERATIVA

La **búsqueda con profundidad iterativa** (o búsqueda primero en profundidad con profundidad iterativa) es una estrategia general, usada a menudo en combinación con la búsqueda primero en profundidad, la cual encuentra el mejor límite de profundidad. Esto se hace aumentando gradualmente el límite (primero 0, después 1, después 2, etcétera) hasta que encontramos un objetivo. Esto ocurrirá cuando el límite de profundidad alcanza d, profundidad del nodo objetivo. Se muestra en la Figura 3.14 el algoritmo. La profundidad iterativa combina las ventajas de la búsqueda primero en profundidad y primero en anchura. En la búsqueda primero en profundidad, sus exigencias de memoria

```
función BÚSQUEDA-PROFUNDIDAD-ITERATIVA(problema) devuelve una solución, o fallo
  entradas: problema, un problema

  para profundidad ← 0 a ∞ hacer
    resultado ← BÚSQUEDA-PROFUNDIDAD-LIMITADA(problema,profundidad)
    si resultado ≠ corte entonces devolver resultado
```

Figura 3.14 Algoritmo de búsqueda de profundidad iterativa, el cual aplica repetidamente la búsqueda de profundidad limitada incrementando el límite. Termina cuando se encuentra una solución o si la búsqueda de profundidad limitada devuelve *fracaso*, significando que no existe solución.

son muy modestas: $O(bd)$ para ser exacto. La búsqueda primero en anchura, es completa cuando el factor de ramificación es finito y óptima cuando el coste del camino es una función que no disminuye con la profundidad del nodo. La Figura 3.15 muestra cuatro iteraciones de la BÚSQUEDA-PROFUNDIDAD-ITERATIVA sobre un árbol binario de búsqueda, donde la solución se encuentra en la cuarta iteración.

La búsqueda de profundidad iterativa puede parecer derrochadora, porque los estados se generan múltiples veces. Pero esto no es muy costoso. La razón es que en un árbol de búsqueda con el mismo (o casi el mismo) factor de ramificación en cada nivel, la mayor parte de los nodos está en el nivel inferior, entonces no importa mucho que los niveles superiores se generen múltiples veces. En una búsqueda de profundidad iterativa, los nodos sobre el nivel inferior (profundidad d) son generados una vez, los anteriores al nivel inferior son generados dos veces, etc., hasta los hijos de la raíz, que son generados d veces. Entonces el número total de nodos generados es

$$N(BPI) = (d)b + (d - 1)b^2 + ... + (1)b^d,$$

que da una complejidad en tiempo de $O(b^d)$. Podemos compararlo con los nodos generados por una búsqueda primero en anchura:

$$N(BPA) = b + b^2 + ... + b^d + (b^{d+1} - b).$$

Notemos que la búsqueda primero en anchura genera algunos nodos en profundidad $d + 1$, mientras que la profundidad iterativa no lo hace. El resultado es que la profundidad iterativa es en realidad más rápida que la búsqueda primero en anchura, a pesar de la generación repetida de estados. Por ejemplo, si $b = 10$ y $d = 5$, los números son

$$N(BPI) = 50 + 400 + 3.000 + 20.000 + 100.000 = 123.450$$
$$N(BPA) = 10 + 100 + 1.000 + 10.000 + 100.000 + 999.990 = 1.111.100$$

En general, la profundidad iterativa es el método de búsqueda no informada preferido cuando hay un espacio grande de búsqueda y no se conoce la profundidad de la solución.

La búsqueda de profundidad iterativa es análoga a la búsqueda primero en anchura en la cual se explora, en cada iteración, una capa completa de nuevos nodos antes de continuar con la siguiente capa. Parecería que vale la pena desarrollar una búsqueda iterativa análoga a la búsqueda de coste uniforme, heredando las garantías de optimización del algoritmo evitando sus exigencias de memoria. La idea es usar límites crecientes de costo del camino en vez de aumentar límites de profundidad. El algoritmo que resulta,

Figura 3.15 Cuatro iteraciones de la búsqueda de profundidad iterativa sobre un árbol binario.

BÚSQUEDA DE LONGITUD ITERATIVA

llamado **búsqueda de longitud iterativa**, se explora en el Ejercicio 3.11. Resulta, lamentablemente, que la longitud iterativa incurre en gastos indirectos sustanciales comparado con la búsqueda de coste uniforme.

Búsqueda bidireccional

La idea de la búsqueda bidireccional es ejecutar dos búsquedas simultáneas: una hacia delante desde el estado inicial y la otra hacia atrás desde el objetivo, parando cuando las dos búsquedas se encuentren en el centro (Figura 3.16). La motivación es que $b^{d/2} + b^{d/2}$ es mucho menor que b^d, o en la figura, el área de los dos círculos pequeños es menor que el área de un círculo grande centrado en el inicio y que alcance al objetivo.

La búsqueda bidireccional se implementa teniendo una o dos búsquedas que comprueban antes de ser expandido si cada nodo está en la frontera del otro árbol de bús-

Figura 3.16 Un esquema de una búsqueda bidireccional que está a punto de tener éxito, cuando una rama desde el nodo inicio encuentra una rama desde el nodo objetivo.

queda; si esto ocurre, se ha encontrado una solución. Por ejemplo, si un problema tiene una solución a profundidad $d = 6$, y en cada dirección se ejecuta la búsqueda primero en anchura, entonces, en el caso peor, las dos búsquedas se encuentran cuando se han expandido todos los nodos excepto uno a profundidad 3. Para $b = 10$, esto significa un total de 22.200 nodos generados, comparado con 11.111.100 para una búsqueda primero en anchura estándar. Verificar que un nodo pertenece al otro árbol de búsqueda se puede hacer en un tiempo constante con una tabla *hash*, así que la complejidad en tiempo de la búsqueda bidireccional es $O(b^{d/2})$. Por lo menos uno de los árboles de búsqueda se debe mantener en memoria para que se pueda hacer la comprobación de pertenencia, de ahí que la complejidad en espacio es también $O(b^{d/2})$. Este requerimiento de espacio es la debilidad más significativa de la búsqueda bidireccional. El algoritmo es completo y óptimo (para costos uniformes) si las búsquedas son primero en anchura; otras combinaciones pueden sacrificar la completitud, optimización, o ambas.

PREDECESORES

La reducción de complejidad en tiempo hace a la búsqueda bidireccional atractiva, pero ¿cómo busca hacia atrás? Esto no es tan fácil como suena. Sean los **predecesores** de un nodo n, $Pred(n)$, todos los nodos que tienen como un sucesor a n. La búsqueda bidireccional requiere que $Pred(n)$ se calcule eficientemente. El caso más fácil es cuando todas las acciones en el espacio de estados son reversibles, así que $Pred(n) = Succ(n)$. Otro caso puede requerir ser ingenioso.

Consideremos la pregunta de qué queremos decir con «el objetivo» en la búsqueda «hacia atrás». Para el 8-puzle y para encontrar un camino en Rumanía, hay solamente un estado objetivo, entonces la búsqueda hacia atrás se parece muchísimo a la búsqueda hacia delante. Si hay varios estados objetivo explícitamente catalogados (por ejemplo, los dos estados objetivo sin suciedad de la Figura 3.3) podemos construir un nuevo estado objetivo ficticio cuyos predecesores inmediatos son todos los estados objetivo reales. Alternativamente, algunos nodos generados redundantes se pueden evitar vien-

do el conjunto de estados objetivo como uno solo, cada uno de cuyos predecesores es también un conjunto de estados (específicamente, el conjunto de estados que tienen a un sucesor en el conjunto de estados objetivo. *Véase* también la Sección 3.6).

El caso más difícil para la búsqueda bidireccional es cuando el test objetivo da sólo una descripción implícita de algún conjunto posiblemente grande de estados objetivo, por ejemplo, todos los estados que satisfacen el test objetivo «jaque mate» en el ajedrez. Una búsqueda hacia atrás necesitaría construir las descripciones de «todos los estados que llevan al jaque mate al mover m_1», etcétera; y esas descripciones tendrían que ser probadas de nuevo con los estados generados en la búsqueda hacia delante. No hay ninguna manera general de hacer esto eficientemente.

Comparación de las estrategias de búsqueda no informada

La Figura 3.17 compara las estrategias de búsqueda en términos de los cuatro criterios de evaluación expuestos en la Sección 3.4.

Criterio	Primero en anchura	Costo uniforme	Primero en profundidad	Profundidad limitada	Profundidad iterativa	Bidireccional (si aplicable)
¿Completa?	Sí[a]	Sí[a,b]	No	No	Sí[a]	Sí[a,d]
Tiempo	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Espacio	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
¿Optimal?	Sí[c]	Sí	No	No	Sí[c]	Sí[c,d]

Figura 3.17 Evaluación de estrategias de búsqueda. b es el factor de ramificación; d es la profundidad de la solución más superficial; m es la máxima profundidad del árbol de búsqueda; ℓ es el límite de profundidad. Los superíndice significan lo siguiente: [a] completa si b es finita; [b] completa si los costos son $\geq \epsilon$ para ϵ positivo; [c] optimal si los costos son iguales; [d] si en ambas direcciones se utiliza la búsqueda primero en anchura.

3.5 Evitar estados repetidos

Hasta este punto, casi hemos ignorado una de las complicaciones más importantes al proceso de búsqueda: la posibilidad de perder tiempo expandiendo estados que ya han sido visitados y expandidos. Para algunos problemas, esta posibilidad nunca aparece; el espacio de estados es un árbol y hay sólo un camino a cada estado. La formulación eficiente del problema de las ocho reinas (donde cada nueva reina se coloca en la columna vacía de más a la izquierda) es eficiente en gran parte a causa de esto (cada estado se puede alcanzar sólo por un camino). Si formulamos el problema de las ocho reinas para poder colocar una reina en cualquier columna, entonces cada estado con n reinas se puede alcanzar por $n!$ caminos diferentes.

Para algunos problemas, la repetición de estados es inevitable. Esto incluye todos los problemas donde las acciones son reversibles, como son los problemas de búsque-

da de rutas y los puzles que deslizan sus piezas. Los árboles de la búsqueda para estos problemas son infinitos, pero si podamos parte de los estados repetidos, podemos cortar el árbol de búsqueda en un tamaño finito, generando sólo la parte del árbol que atraviesa el grafo del espacio de estados. Considerando solamente el árbol de búsqueda hasta una profundidad fija, es fácil encontrar casos donde la eliminación de estados repetidos produce una reducción exponencial del coste de la búsqueda. En el caso extremo, un espacio de estados de tamaño $d + 1$ (Figura 3.18(a)) se convierte en un árbol con 2^d hojas (Figura 3.18(b)). Un ejemplo más realista es la **rejilla rectangular**, como se ilustra en la Figura 3.18(c). Sobre una rejilla, cada estado tiene cuatro sucesores, entonces el árbol de búsqueda, incluyendo estados repetidos, tiene 4^d hojas; pero hay sólo $2d^2$ estados distintos en d pasos desde cualquier estado. Para $d = 20$, significa aproximadamente un billón de nodos, pero aproximadamente 800 estados distintos.

REJILLA RECTANGULAR

Entonces, si el algoritmo no detecta los estados repetidos, éstos pueden provocar que un problema resoluble llegue a ser irresoluble. La detección por lo general significa la comparación del nodo a expandir con aquellos que han sido ya expandidos; si se encuentra un emparejamiento, entonces el algoritmo ha descubierto dos caminos al mismo estado y puede desechar uno de ellos.

Para la búsqueda primero en profundidad, los únicos nodos en memoria son aquellos del camino desde la raíz hasta el nodo actual. La comparación de estos nodos permite al algoritmo descubrir los caminos que forman ciclos y que pueden eliminarse inmediatamente. Esto está bien para asegurar que espacios de estados finitos no hagan árboles de búsqueda infinitos debido a los ciclos; lamentablemente, esto no evita la proliferación exponencial de caminos que no forman ciclos, en problemas como los de la Figura 3.18. El único modo de evitar éstos es guardar más nodos en la memoria. Hay una compensación fundamental entre el espacio y el tiempo. *Los algoritmos que olvidan su historia están condenados a repetirla.*

Figura 3.18 Espacios de estados que genera un árbol de búsqueda exponencialmente más grande. (a) Un espacio de estados en el cual hay dos acciones posibles que conducen de A a B, dos de B a C, etcétera. El espacio de estados contiene $d + 1$ estados, donde d es la profundidad máxima. (b) Correspondiente árbol de búsqueda, que tiene 2^d ramas correspondientes a 2^d caminos en el espacio. (c) Un espacio rejilla rectangular. Se muestran en gris los estados dentro de dos pasos desde el estado inicial (A).

LISTA CERRADA

LISTA ABIERTA

Si un algoritmo recuerda cada estado que ha visitado, entonces puede verse como la exploración directamente del grafo de espacio de estados. Podemos modificar el algoritmo general de BÚSQUEDA-ÁRBOLES para incluir una estructura de datos llamada **lista cerrada**, que almacene cada nodo expandido. (A veces se llama a la frontera de nodos no expandidos **lista abierta**.) Si el nodo actual se empareja con un nodo de la lista cerrada, se elimina en vez de expandirlo. Al nuevo algoritmo se le llama BÚSQUEDA-GRAFOS. Sobre problemas con muchos estados repetidos, la BÚSQUEDA-GRAFOS es mucho más eficiente que la BÚSQUEDA-ÁRBOLES. Los requerimientos en tiempo y espacio, en el caso peor, son proporcionales al tamaño del espacio de estados. Esto puede ser mucho más pequeño que $O(b^d)$.

La optimización para la búsqueda en grafos es una cuestión difícil. Dijimos antes que cuando se detecta un estado repetido, el algoritmo ha encontrado dos caminos al mismo estado. El algoritmo de BÚSQUEDA-GRAFOS de la Figura 3.19 siempre desecha el camino recién descubierto; obviamente, si el camino recién descubierto es más corto que el original, la BÚSQUEDA-GRAFOS podría omitir una solución óptima. Afortunadamente, podemos mostrar (Ejercicio 3.12) que esto no puede pasar cuando utilizamos la búsqueda de coste uniforme o la búsqueda primero en anchura con costos constantes; de ahí, que estas dos estrategias óptimas de búsqueda en árboles son también estrategias óptimas de búsqueda en grafos. La búsqueda con profundidad iterativa, por otra parte, utiliza la expansión primero en profundidad y fácilmente puede seguir un camino subóptimo a un nodo antes de encontrar el óptimo. De ahí, la búsqueda en grafos de profundidad iterativa tiene que comprobar si un camino recién descubierto a un nodo es mejor que el original, y si es así, podría tener que revisar las profundidades y los costos del camino de los descendientes de ese nodo.

Notemos que el uso de una lista cerrada significa que la búsqueda primero en profundidad y la búsqueda en profundidad iterativa tienen unos requerimientos lineales en espacio. Como el algoritmo de BÚSQUEDA-GRAFOS mantiene cada nodo en memoria, algunas búsquedas son irrealizables debido a limitaciones de memoria.

```
función BÚSQUEDA-GRAFOS(problema,frontera) devuelve una solución, o fallo

  cerrado ← conjunto vacío
  frontera ← INSERTAR(HACER-NODO(ESTADO-INICIAL[problema]),frontera)
  bucle hacer
    si VACIA?(frontera) entonces devolver fallo
    nodo ← BORRAR-PRIMERO(frontera)
    si TEST-OBJETIVO[problema](ESTADO[nodo]) entonces devolver SOLUCIÓN(nodo)
    si ESTADO[nodo] no está en cerrado entonces
      añadir ESTADO[nodo] a cerrado
      frontera ← INSERTAR-TODO(EXPANDIR(nodo,problema),frontera)
```

Figura 3.19 Algoritmo general de búsqueda en grafos. El conjunto cerrado puede implementarse como una tabla *hash* para permitir la comprobación eficiente de estados repetidos. Este algoritmo supone que el primer camino a un estado *s* es el más barato (*véase* el texto).

3.6 Búsqueda con información parcial

En la Sección 3.3 asumimos que el entorno es totalmente observable y determinista y que el agente conoce cuáles son los efectos de cada acción. Por lo tanto, el agente puede calcular exactamente cuál es el estado resultado de cualquier secuencia de acciones y siempre sabe en qué estado está. Su percepción no proporciona ninguna nueva información después de cada acción. ¿Qué pasa cuando el conocimiento de los estados o acciones es incompleto? Encontramos que diversos tipos de incompletitud conducen a tres tipos de problemas distintos:

1. **Problemas sin sensores** (también llamados **problemas conformados**): si el agente no tiene ningún sensor, entonces (por lo que sabe) podría estar en uno de los posibles estados iniciales, y cada acción por lo tanto podría conducir a uno de los posibles estados sucesores.
2. **Problemas de contingencia**: si el entorno es parcialmente observable o si las acciones son inciertas, entonces las percepciones del agente proporcionan *nueva* información después de cada acción. Cada percepción posible define una contingencia que debe de planearse. A un problema se le llama entre **adversarios** si la incertidumbre está causada por las acciones de otro agente.
3. **Problemas de exploración**: cuando se desconocen los estados y las acciones del entorno, el agente debe actuar para descubrirlos. Los problemas de exploración pueden verse como un caso extremo de problemas de contingencia.

Como ejemplo, utilizaremos el entorno del mundo de la aspiradora. Recuerde que el espacio de estados tiene ocho estados, como se muestra en la Figura 3.20. Hay tres acciones (*Izquierda, Derecha* y *Aspirar*) y el objetivo es limpiar toda la suciedad (estados 7 y 8). Si el entorno es observable, determinista, y completamente conocido, entonces el problema es trivialmente resoluble por cualquiera de los algoritmos que hemos descrito. Por

Figura 3.20 Los ocho posibles estados del mundo de la aspiradora.

ejemplo, si el estado inicial es 5, entonces la secuencia de acciones [*Derecha, Aspirar*] alcanzará un estado objetivo, 8. El resto de esta sección trata con las versiones sin sensores y de contingencia del problema. Los problemas de exploración se tratan en la Sección 4.5, los problemas entre adversarios en el Capítulo 6.

Problemas sin sensores

Supongamos que el agente de la aspiradora conoce todos los efectos de sus acciones, pero no tiene ningún sensor. Entonces sólo sabe que su estado inicial es uno del conjunto {1, 2, 3, 4, 5, 6, 7, 8}. Quizá supongamos que el agente está desesperado, pero de hecho puede hacerlo bastante bien. Como conoce lo que hacen sus acciones, puede, por ejemplo, calcular que la acción *Derecha* produce uno de los estados {2, 4, 6, 8}, y la secuencia de acción [*Derecha, Aspirar*] siempre terminará en uno de los estados {4, 8}. Finalmente, la secuencia [*Derecha, Aspirar, Izquierda, Aspirar*] garantiza alcanzar el estado objetivo 7 sea cual sea el estado inicio. Decimos que el agente puede **coaccionar** al mundo en el estado 7, incluso cuando no sepa dónde comenzó. Resumiendo: cuando el mundo no es completamente observable, el agente debe decidir sobre los *conjuntos* de estados que podría poner, más que por estados simples. Llamamos a cada conjunto de estados un **estado de creencia**, representando la creencia actual del agente con los estados posibles físicos en los que podría estar. (En un ambiente totalmente observable, cada estado de creencia contiene un estado físico.)

COACCIÓN

ESTADO DE CREENCIA

Para resolver problemas sin sensores, buscamos en el espacio de estados de creencia más que en los estados físicos. El estado inicial es un estado de creencia, y cada acción aplica un estado de creencia en otro estado de creencia. Una acción se aplica a un estado de creencia uniendo los resultados de aplicar la acción a cada estado físico del estado de creencia. Un camino une varios estados de creencia, y una solución es ahora un camino que conduce a un estado de creencia, *todos de cuyos miembros* son estados objetivo. La Figura 3.21 muestra el espacio de estados de creencia accesible para el mundo determinista de la aspiradora sin sensores. Hay sólo 12 estados de creencia accesibles, pero el espacio de estados de creencia entero contiene todo conjunto posible de estados físicos, por ejemplo, $2^8 = 256$ estados de creencia. En general, si el espacio de estados físico tiene S estados, el espacio de estados de creencia tiene 2^S estados de creencia.

Nuestra discusión hasta ahora de problemas sin sensores ha supuesto acciones deterministas, pero el análisis es esencialmente el mismo si el entorno es no determinista, es decir, si las acciones pueden tener varios resultados posibles. La razón es que, en ausencia de sensores, el agente no tiene ningún modo de decir qué resultado ocurrió en realidad, así que varios resultados posibles son estados físicos adicionales en el estado de creencia sucesor. Por ejemplo, supongamos que el entorno obedece a la ley de Murphy: la llamada acción de *Aspirar a veces* deposita suciedad en la alfombra *pero sólo si no ha ninguna suciedad allí*[6]. Entonces, si *Aspirar* se aplica al estado físico 4 (mirar la Figura 3.20), hay dos resultados posibles: los estados 2 y 4. Aplicado al estado de

[6] Suponemos que la mayoría de los lectores afrontan problemas similares y pueden simpatizar con nuestro agente. Nos disculpamos a los dueños de los aparatos electrodomésticos modernos y eficientes que no pueden aprovecharse de este dispositivo pedagógico.

Figura 3.21 La parte accesible del espacio de estados de creencia para el mundo determinista de la aspiradora sin sensores. Cada caja sombreada corresponde a un estado de creencia simple. Desde cualquier punto, el agente está en un estado de creencia particular pero no sabe en qué estado físico está. El estado de creencia inicial (con ignorancia completa) es la caja central superior. Las acciones se representan por arcos etiquetados. Los autoarcos se omiten por claridad.

creencia inicial, {1, 2, 3, 4, 5, 6, 7 ,8}, *Aspirar* conduce al estado de creencia que es la unión de los conjuntos resultados para los ocho estados físicos. Calculándolos, encontramos que el nuevo estado de creencia es {1, 2, 3, 4, 5, 6, 7, 8}. ¡Así, para un agente sin sensores en el mundo de la ley de Murphy, la acción *Aspirar* deja el estado de creencia inalterado! De hecho, el problema es no resoluble. (*Véase* el Ejercicio 3.18.) Por intuición, la razón es que el agente no puede distinguir si el cuadrado actual está sucio y de ahí que no puede distinguir si la acción *Aspirar* lo limpiará o creará más suciedad.

Problemas de contingencia

PROBLEMA DE CONTINGENCIA

Cuando el entorno es tal que el agente puede obtener nueva información de sus sensores después de su actuación, el agente afronta **problemas de contingencia**. La solución en un problema de contingencia a menudo toma la forma de un árbol, donde cada rama se

puede seleccionar según la percepción recibida en ese punto del árbol. Por ejemplo, supongamos que el agente está en el mundo de la ley de Murphy y que tiene un sensor de posición y un sensor de suciedad local, pero no tiene ningún sensor capaz de detectar la suciedad en otros cuadrados. Así, la percepción [*I*, *Sucio*] significa que el agente está en uno de los estados {1, 3}. El agente podría formular la secuencia de acciones [*Aspirar*, *Derecha*, *Aspirar*]. *Aspirar* cambiaría el estado al {5, 7}, y el mover a la derecha debería entonces cambiar el estado al {6, 8}. La ejecución de la acción final de *Aspirar* en el estado 6 nos lleva al estado 8, un objetivo, pero la ejecución en el estado 8 podría llevarnos para atrás al estado 6 (según la ley de Murphy), en el caso de que el plan falle.

Examinando el espacio de estados de creencia para esta versión del problema, fácilmente puede determinarse que ninguna secuencia de acciones rígida garantiza una solución a este problema. Hay, sin embargo, una solución si no insistimos en una secuencia de acciones *rígida*:

[*Aspirar*, *Derecha*, **si** [*D,Suciedad*] **entonces** *Aspirar*].

Esto amplía el espacio de soluciones para incluir la posibilidad de seleccionar acciones basadas en contingencias que surgen durante la ejecución. Muchos problemas en el mundo real, físico, son problemas de contingencia, porque la predicción exacta es imposible. Por esta razón, todas las personas mantienen sus ojos abiertos mientras andan o conducen.

Los problemas de contingencia *a veces* permiten soluciones puramente secuenciales. Por ejemplo, considere el mundo de la ley de Murphy *totalmente observable*. Las contingencias surgen si el agente realiza una acción *Aspirar* en un cuadrado limpio, porque la suciedad podría o no ser depositada en el cuadrado. Mientras el agente nunca haga esto, no surge ninguna contingencia y hay una solución secuencial desde cada estado inicial (Ejercicio 3.18).

Los algoritmos para problemas de contingencia son más complejos que los algoritmos estándar de búsqueda de este capítulo; ellos serán tratados en el Capítulo 12. Los problemas de contingencia también se prestan a un diseño de agente algo diferente, en el cual el agente puede actuar *antes* de que haya encontrado un plan garantizado. Esto es útil porque más que considerar por adelantado cada posible contingencia que *podría* surgir durante la ejecución, es a menudo mejor comenzar a actuar y ver qué contingencias surgen realmente. Entonces el agente puede seguir resolviendo el problema, teniendo en cuenta la información adicional. Este tipo de **intercalar** la búsqueda y la ejecución es también útil para problemas de exploración (*véase* la Sección 4.5) y para juegos (*véase* el Capítulo 6).

INTERCALAR

3.7 Resumen

Este capítulo ha introducido métodos en los que un agente puede seleccionar acciones en los ambientes deterministas, observables, estáticos y completamente conocidos. En tales casos, el agente puede construir secuencias de acciones que alcanzan sus objetivos; a este proceso se le llama **búsqueda**.

- Antes de que un agente pueda comenzar la búsqueda de soluciones, debe formular un objetivo y luego usar dicho objetivo para formular un **problema**.

- Un problema consiste en cuatro partes: el **estado inicial**, un conjunto de **acciones**, una función para el **test objetivo**, y una función de **costo del camino**. El entorno del problema se representa por un **espacio de estados**. Un **camino** por el espacio de estados desde el estado inicial a un estado objetivo es una **solución**.
- Un algoritmo sencillo y general de BÚSQUEDA-ÁRBOL puede usarse para resolver cualquier problema; las variantes específicas del algoritmo incorporan estrategias diferentes.
- Los algoritmos de búsqueda se juzgan sobre la base de **completitud**, **optimización**, **complejidad en tiempo** y **complejidad en espacio**. La complejidad depende de b, factor de ramificación en el espacio de estados, y d, profundidad de la solución más superficial.
- La **búsqueda primero en anchura** selecciona para su expansión el nodo no expandido más superficial en el árbol de búsqueda. Es completo, óptimo para costos unidad, y tiene la complejidad en tiempo y en espacio de $O(b^d)$. La complejidad en espacio lo hace poco práctico en la mayor parte de casos. La **búsqueda de coste uniforme** es similar a la búsqueda primero en anchura pero expande el nodo con el costo más pequeño del camino, $g(n)$. Es completo y óptimo si el costo de cada paso excede de una cota positiva ϵ.
- La **búsqueda primero en profundidad** selecciona para la expansión el nodo no expandido más profundo en el árbol de búsqueda. No es ni completo, ni óptimo, y tiene la complejidad en tiempo de $O(b^m)$ y la complejidad en espacio de $O(bm)$, donde m es la profundidad máxima de cualquier camino en el espacio de estados.
- La **búsqueda de profundidad limitada** impone un límite de profundidad fijo a una búsqueda primero en profundidad.
- La **búsqueda de profundidad iterativa** llama a la búsqueda de profundidad limitada aumentando este límite hasta que se encuentre un objetivo. Es completo, óptimo para costos unidad, y tiene la complejidad en tiempo de $O(b^d)$ y la complejidad en espacio de $O(bd)$.
- La **búsqueda bidireccional** puede reducir enormemente la complejidad en tiempo, pero no es siempre aplicable y puede requerir demasiado espacio.
- Cuando el espacio de estados es un grafo más que un árbol, puede valer la pena comprobar si hay estados repetidos en el árbol de búsqueda. El algoritmo de BÚSQUEDA-GRAFOS elimina todos los estados duplicados.
- Cuando el ambiente es parcialmente observable, el agente puede aplicar algoritmos de búsqueda en el espacio de **estados de creencia**, o los conjuntos de estados posibles en los cuales el agente podría estar. En algunos casos, se puede construir una sencilla secuencia solución; en otros casos, el agente necesita de un **plan de contingencia** para manejar las circunstancias desconocidas que puedan surgir.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Casi todos los problemas de búsqueda en espacio de estados analizados en este capítulo tienen una larga historia en la literatura y son menos triviales de lo que parecían. El problema de los misioneros y caníbales, utilizado en el ejercicio 3.9, fue analizado con

detalle por Amarel (1968). Antes fue considerado en IA por Simon y Newel (1961), y en Investigación Operativa por Bellman y Dreyfus (1962). Estudios como estos y el trabajo de Newell y Simon sobre el Lógico Teórico (1957) y GPS (1961) provocaron el establecimiento de los algoritmos de búsqueda como las armas principales de los investigadores de IA en 1960 y el establecimiento de la resolución de problemas como la tarea principal de la IA. Desafortunadamente, muy pocos trabajos se han hecho para automatizar los pasos para la formulación de los problemas. Un tratamiento más reciente de la representación y abstracción del problema, incluyendo programas de IA que los llevan a cabo (en parte), está descrito en Knoblock (1990).

El 8-puzle es el primo más pequeño del 15-puzle, que fue inventado por el famoso americano diseñador de juegos Sam Loyd (1959) en la década de 1870. El 15-puzle rápidamente alcanzó una inmensa popularidad en Estados Unidos, comparable con la más reciente causada por el Cubo de Rubik. También rápidamente atrajo la atención de matemáticos (Johnson y Story, 1879; Tait, 1880). Los editores de la *Revista Americana de Matemáticas* indicaron que «durante las últimas semanas el 15-puzle ha ido creciendo en interés ante el público americano, y puede decirse con seguridad haber captado la atención de nueve de cada diez personas, de todas las edades y condiciones de la comunidad. Pero esto no ha inducido a los editores a incluir artículos sobre tal tema en la *Revista Americana de Matemáticas*, pero para el hecho que…» (sigue un resumen de interés matemático del 15-puzle). Un análisis exhaustivo del 8-puzle fue realizado por Schofield (1967) con la ayuda de un computador. Ratner y Warmuth (1986) mostraron que la versión general $n \times n$ del 15-puzle pertenece a la clase de problemas NP-completos.

El problema de las 8-reinas fue publicado de manera anónima en la revista alemana de ajedrez *Schach* en 1848; más tarde fue atribuido a Max Bezzel. Fue republicado en 1850 y en aquel tiempo llamó la atención del matemático eminente Carl Friedrich Gauss, que intentó enumerar todas las soluciones posibles, pero encontró sólo 72. Nauck publicó más tarde en 1850 las 92 soluciones. Netto (1901) generalizó el problema a n reinas, y Abramson y Yung (1989) encontraron un algoritmo de orden $O(n)$.

Cada uno de los problemas de búsqueda del mundo real, catalogados en el capítulo, han sido el tema de mucho esfuerzo de investigación. Los métodos para seleccionar vuelos óptimos de líneas aéreas siguen estando mayoritariamente patentados, pero Carl de Marcken (personal de comunicaciones) ha demostrado que las restricciones y el precio de los billetes de las líneas aéreas lo convierten en algo tan enrevesado que el problema de seleccionar un vuelo óptimo es formalmente *indecidible*. El problema del viajante de comercio es un problema combinatorio estándar en informática teórica (Lawler, 1985; Lawler *et al*., 1992). Karp (1972) demostró que el PVC es NP-duro, pero se desarrollaron métodos aproximados heurísticos efectivos (Lin y Kernighan, 1973). Arora (1998) inventó un esquema aproximado polinomial completo para los PVC Euclídeos. Shahookar y Mazumder (1991) inspeccionaron los métodos para la distribución VLSI, y muchos trabajos de optimización de distribuciones aparecen en las revistas de VLSI. La navegación robótica y problemas de ensamblaje se discuten en el Capítulo 25.

Los algoritmos de búsqueda no informada para resolver un problema son un tema central de la informática clásica (Horowitz y Sahni, 1978) y de la investigación operativa (Dreyfus, 1969); Deo y Pang (1984) y Gallo y Pallottino (1988) dan revisiones más

recientes. La búsqueda primero en anchura fue formulada por Moore (1959) para resolver laberintos. El método de **programación dinámica** (Bellman y Dreyfus, 1962), que sistemáticamente registra soluciones para todos los sub-problemas de longitudes crecientes, puede verse como una forma de búsqueda primero en anchura sobre grafos. El algoritmo de camino más corto entre dos puntos de Dijkstra (1959) es el origen de búsqueda de coste uniforme.

Una versión de profundidad iterativa, diseñada para hacer eficiente el uso del reloj en el ajedrez, fue utilizada por Slate y Atkin (1977) en el programa de juego AJEDREZ 4.5, pero la aplicación a la búsqueda del camino más corto en grafos se debe a Korf (1985a). La búsqueda bidireccional, que fue presentada por Pohl (1969,1971), también puede ser muy eficaz en algunos casos.

Ambientes parcialmente observables y no deterministas no han sido estudiados en gran profundidad dentro de la resolución de problemas. Algunas cuestiones de eficacia en la búsqueda en estados de creencia han sido investigadas por Genesereth y Nourbakhsh (1993). Koening y Simmons (1998) estudió la navegación del robot desde una posición desconocida inicial, y Erdmann y Mason (1988) estudió el problema de la manipulación robótica sin sensores, utilizando una forma continua de búsqueda en estados de creencia. La búsqueda de contingencia ha sido estudiada dentro del subcampo de la planificación (*véase* el Capítulo 12). Principalmente, planificar y actuar con información incierta se ha manejado utilizando las herramientas de probabilidad y la teoría de decisión (*véase* el Capítulo 17).

Los libros de texto de Nilsson (1971,1980) son buenas fuentes bibliográficas generales sobre algoritmos clásicos de búsqueda. Una revisión comprensiva y más actualizada puede encontrarse en Korf (1988). Los artículos sobre nuevos algoritmos de búsqueda (que, notablemente, se siguen descubriendo) aparecen en revistas como *Artificial Intelligence*.

EJERCICIOS

3.1 Defina con sus propias palabras los siguientes términos: estado, espacio de estados, árbol de búsqueda, nodo de búsqueda, objetivo, acción, función sucesor, y factor de ramificación.

3.2 Explique por qué la formulación del problema debe seguir a la formulación del objetivo.

3.3 Supongamos que ACCIONES-LEGALES(s) denota el conjunto de acciones que son legales en el estado s, y RESULTADO(a,s) denota el estado que resulta de la realización de una acción legal a a un estado s. Defina FUNCIÓN-SUCESOR en términos ACCIONES-LEGALES y RESULTADO, y viceversa.

3.4 Demuestre que los estados del 8-puzle están divididos en dos conjuntos disjuntos, tales que ningún estado en un conjunto puede transformarse en un estado del otro conjunto por medio de un número de movimientos. (*Consejo*: *véase* Berlekamp *et al.* (1982)). Invente un procedimiento que nos diga en qué clase está un estado dado, y explique por qué esto es bueno cuando generamos estados aleatorios.

3.5 Consideremos el problema de las 8-reinas usando la formulación «eficiente» incremental de la página 75. Explique por qué el tamaño del espacio de estados es al menos $\sqrt[3]{n!}$ y estime el valor más grande para n para el cual es factible la exploración exhaustiva. (*Consejo*: saque una cota inferior para el factor de ramificación considerando el número máximo de cuadrados que una reina puede atacar en cualquier columna.)

3.6 ¿Conduce siempre un espacio de estados finito a un árbol de búsqueda finito? ¿Cómo un espacio de estados finito es un árbol? ¿Puede ser más preciso sobre qué tipos de espacios de estados siempre conducen a árboles de búsqueda finito? (Adaptado de Bender, 1996.)

3.7 Defina el estado inicial, test objetivo, función sucesor, y función costo para cada uno de los siguientes casos. Escoja una formulación que sea suficientemente precisa para ser implementada.

- ***a***) Coloree un mapa plano utilizando sólo cuatro colores, de tal modo que dos regiones adyacentes no tengan el mismo color.
- ***b***) Un mono de tres pies de alto está en una habitación en donde algunos plátanos están suspendidos del techo de ocho pies de alto. Le gustaría conseguir los plátanos. La habitación contiene dos cajas apilables, móviles y escalables de tres pies de alto.
- ***c***) Tiene un programa que da como salida el mensaje «registro de entrada ilegal» cuando introducimos un cierto archivo de registros de entrada. Sabe que el tratamiento de cada registro es independiente de otros registros. Quiere descubrir que es ilegal.
- ***d***) Tiene tres jarros, con capacidades 12 galones, ocho galones, y tres galones, y un grifo de agua. Usted puede llenar los jarros o vaciarlos de uno a otro o en el suelo. Tiene que obtener exactamente un galón.

3.8 Considere un espacio de estados donde el estado comienzo es el número 1 y la función sucesor para el estado n devuelve 2 estados, los números $2n$ y $2n + 1$.

- ***a***) Dibuje el trozo del espacio de estados para los estados del 1 al 15.
- ***b***) Supongamos que el estado objetivo es el 11. Enumere el orden en el que serán visitados los nodos por la búsqueda primero en anchura, búsqueda primero en profundidad con límite tres, y la búsqueda de profundidad iterativa.
- ***c***) ¿Será apropiada la búsqueda bidireccional para este problema? Si es así, describa con detalle cómo trabajaría.
- ***d***) ¿Qué es el factor de ramificación en cada dirección de la búsqueda bidireccional?
- ***e***) ¿La respuesta (c) sugiere una nueva formulación del problema que permitiría resolver el problema de salir del estado 1 para conseguir un estado objetivo dado, con casi ninguna búsqueda?

3.9 El problema de los **misioneros y caníbales** en general se forma como sigue. tres misioneros y tres caníbales están en un lado de un río, con un barco que puede sostener a una o dos personas. Encuentre un modo de conseguir que todos estén en el otro lado, sin dejar alguna vez a un grupo de misioneros en un lugar excedido en número por los

caníbales. Este problema es famoso en IA porque fue el tema del primer trabajo que aproximó una formulación de problema de un punto de vista analítico (Amarel, 1968).

a) Formule el problema de forma precisa, haciendo sólo las distinciones necesarias para asegurar una solución válida. Dibujar un diagrama del espacio de estados completo.

b) Implemente y resuelva el problema de manera óptima utilizando un algoritmo apropiado de búsqueda. ¿Es una buena idea comprobar los estados repetidos?

c) ¿Por qué cree que la gente utiliza mucho tiempo para resolver este puzle, dado que el espacio de estados es tan simple?

3.10 Implemente dos versiones de la función sucesor para el 8-puzle: uno que genere todos los sucesores a la vez copiando y editando la estructura de datos del 8-puzle, y otro que genere un nuevo sucesor cada vez, llamando y modificando el estado padre directamente (haciendo las modificaciones necesarias). Escriba versiones de la búsqueda primero en profundidad con profundidad iterativa que use estas funciones y compare sus rendimientos.

3.11 En la página 89, mencionamos la **búsqueda de longitud iterativa**, una búsqueda análoga a la de costo uniforme iterativa. La idea es usar incrementos de los límites en el costo del camino. Si se genera un nodo cuyo costo del camino excede el límite actual, se descarta inmediatamente. Para cada nueva iteración, el límite se pone al coste más bajo del camino de cualquier nodo descartado en la iteración anterior.

a) Muestre que este algoritmo es óptimo para costos de camino generales.

b) Considere un árbol uniforme con factor de ramificación b, profundidad de solución d, y costos unidad. ¿Cuántas iteraciones requerirá la longitud iterativa?

c) Ahora considere los costos en el rango continuo [0,1] con un coste mínimo positivo ϵ. ¿Cuántas iteraciones requieren en el peor caso?

d) Implemente el algoritmo y aplíquelo a los casos de los problemas del 8-puzle y del viajante de comercio. Compare el funcionamiento del algoritmo con la búsqueda de costo uniforme, y comente sus resultados.

3.12 Demuestre que la búsqueda de costo uniforme y la búsqueda primero en anchura con costos constantes son óptimas cuando se utiliza con el algoritmo de BÚSQUEDA-GRAFOS. Muestre un espacio de estados con costos constantes en el cual la BÚSQUEDA-GRAFOS, utilizando profundidad iterativa, encuentre una solución subóptima.

3.13 Describa un espacio de estados en el cual la búsqueda de profundidad iterativa funcione mucho peor que la búsqueda primero en profundidad (por ejemplo, $O(n^2)$ contra $O(n)$).

3.14 Escriba un programa que tome como entrada dos URLs de páginas Web y encuentre un camino de links de una a la otra. ¿Cuál es una estrategia apropiada de búsqueda? ¿La búsqueda bidireccional es una idea buena? ¿Podría usarse un motor de búsqueda para implementar una función predecesor?

3.15 Considere el problema de encontrar el camino más corto entre dos puntos sobre un plano que tiene obstáculos poligonales convexos como los que se muestran en la Figura 3.22. Esto es una idealización del problema que tiene un robot para navegar en un entorno muy concurrido.

Figura 3.22 Una escena con obstáculos poligonales.

a) Suponga que el espacio de estados consiste en todas las posiciones (x, y) en el plano. ¿Cuántos estados hay? ¿Cuántos caminos hay al objetivo?

b) Explique brevemente por qué el camino más corto desde un vértice de un polígono a cualquier otro debe consistir en segmentos en línea recta que unen algunos vértices de los polígonos. Defina un espacio de estados bueno. ¿Cómo de grande es este espacio de estados?

c) Defina las funciones necesarias para implementar el problema de búsqueda, incluyendo una función sucesor que toma un vértice como entrada y devuelve el conjunto de vértices que pueden alcanzarse en línea recta desde el vértice dado. (No olvide los vecinos sobre el mismo polígono.) Utilice la distancia en línea recta para la función heurística.

d) Aplique uno o varios de los algoritmos de este capítulo para resolver un conjunto de problemas en el dominio, y comente su funcionamiento.

3.16 Podemos girar el problema de la navegación del Ejercicio 3.15 en un entorno como el siguiente:

- La percepción será una lista de posiciones, *relativas al agente*, de vértices visibles. ¡La percepción *no* incluye la posición del robot! El robot debe aprender su propia posición en el mapa; por ahora, puede suponer que cada posición tiene una «vista» diferente.
- Cada acción será un vector describiendo un camino en línea recta a seguir. Si el camino está libre, la acción tiene éxito; en otro caso, el robot se para en el punto donde el camino cruza un obstáculo. Si el agente devuelve un vector de movimiento cero y está en el objetivo (que está fijado y conocido), entonces el entorno debería colocar al agente en una posición aleatoria (no dentro de un obstáculo).
- La medida de rendimiento carga al agente, un punto por cada unidad de distancia atravesada y concede 1.000 puntos cada vez que se alcance el objetivo.

a) Implemente este entorno y un agente de resolución de problemas. El agente tendrá que formular un nuevo problema después de la colocación en otro lugar, que implicará el descubrimiento de su posición actual.

b) Documente el funcionamiento de su agente (teniendo que generar su agente el comentario conveniente de cómo se mueve a su alrededor) y el informe de su funcionamiento después de más de 100 episodios.

c) Modifique el entorno de modo que el 30 por ciento de veces el agente termine en un destino no planeado (escogido al azar de otros vértices visibles, o que no haya ningún movimiento). Esto es un modelo ordinario de los errores de movimiento de un robot real. Modifique al agente de modo que cuando se descubra tal error, averigüe dónde está y luego construya un plan para regresar donde estaba y resumir el viejo plan. ¡Recuerde que a veces recuperarse hacia donde estaba también podría fallar! Muestre un ejemplo satisfactorio del agente de modo que después de dos errores de movimientos sucesivos, todavía alcance el objetivo.

d) Ahora intente dos esquemas de recuperación diferentes después de un error: (1) diríjase al vértice más cercano sobre la ruta original; y (2) plantee una nueva ruta al objetivo desde la nueva posición. Compare el funcionamiento de estos tres esquemas de recuperación. ¿La inclusión de costos de la búsqueda afecta la comparación?

e) Ahora suponga que hay posiciones de las cuales la vista es idéntica (por ejemplo, suponga que el mundo es una rejilla con obstáculos cuadrados). ¿Qué tipo de problema afronta ahora el agente? ¿A qué se parecen las soluciones?

3.17 En la página 71, dijimos que no consideraríamos problemas con caminos de costos negativos. En este ejercicio, lo exploramos con más profundidad.

a) Suponga que las acciones pueden tener costos negativos arbitrariamente grandes; explique por qué esta posibilidad forzaría a cualquier algoritmo óptimo a explorar entero el espacio de estados.

b) ¿Ayuda si insistimos que los costos puedan ser mayores o iguales a alguna constante negativa c? Considere tanto árboles como grafos.

c) Suponga que hay un conjunto de operadores que forman un ciclo, de modo que la ejecución del conjunto en algún orden no cause ningún cambio neto al estado. Si todos estos operadores tienen el coste negativo, ¿qué implica sobre el comportamiento óptimo para un agente en tal entorno?

d) Fácilmente pueden imaginarse operadores con alto coste negativo, incluso sobre dominios como la búsqueda de rutas. Por ejemplo, algunos caminos podrían tener un paisaje hermoso cuanto más lejano esté y pese más que los gastos normales en términos de tiempo y combustible. Explique, en términos exactos, dentro del contexto del espacio de estados de búsqueda, por qué la gente no conduce sobre ciclos pintorescos indefinidamente, y explique cómo definir el espacio de estados y operadores para la búsqueda de rutas de modo que agentes artificiales también puedan evitar la formación de ciclos.

e) ¿Puede usted pensar en un dominio real en el cual los costos son la causa de la formación de ciclos?

3.18 Considere el mundo de la aspiradora de dos posiciones sin sensores conforme a la ley de Murphy. Dibuje el espacio de estados de creencia accesible desde el estado de creencia inicial $\{1, 2, 3, 4, 5, 6, 7, 8\}$, y explique por qué el problema es irresoluble. Mues-

tre también que si el mundo es totalmente observable, entonces hay una secuencia solución para cada estado inicial posible.

3.19 Considere el problema del mundo de la aspiradora definido en la Figura 2.2

a) ¿Cuál de los algoritmos definidos en este capítulo parece apropiado para este problema? ¿Debería el algoritmo comprobar los estados repetidos?

b) Aplique el algoritmo escogido para obtener una secuencia de acciones óptima para un mundo de 3×3 cuyo estado inicial tiene la suciedad en los tres cuadrados superiores y el agente está en el centro.

c) Construya un agente de búsqueda para el mundo de la aspiradora, y evalúe su rendimiento en un conjunto de 3×3 con probabilidad de 0,2 de suciedad en cada cuadrado.

d) Compare su mejor agente de búsqueda con un agente reactivo sencillo aleatorio que aspira si hay suciedad y en otro caso se mueve aleatoriamente.

e) Considere lo que sucedería si el mundo se ampliase a $n \times n$. ¿Cómo se modificaría el rendimiento del agente de búsqueda y del agente reactivo variando el n?

4 Búsqueda informada y exploración

En donde veremos cómo la información sobre el espacio de estados puede impedir a los algoritmos cometer un error en la oscuridad.

El Capítulo 3 mostró que las estrategias de búsqueda no informadas pueden encontrar soluciones en problemas generando sistemáticamente nuevos estados y probándolos con el objetivo. Lamentablemente, estas estrategias son increíblemente ineficientes en la mayoría de casos. Este capítulo muestra cómo una estrategia de búsqueda informada (la que utiliza el conocimiento específico del problema) puede encontrar soluciones de una manera más eficiente. La Sección 4.1 describe las versiones informadas de los algoritmos del Capítulo 3, y la Sección 4.2 explica cómo se puede obtener la información específica necesaria del problema. Las Secciones 4.3 y 4.4 cubren los algoritmos que realizan la **búsqueda** puramente **local** en el espacio de estados, evaluando y modificando uno o varios estados más que explorando sistemáticamente los caminos desde un estado inicial. Estos algoritmos son adecuados para problemas en los cuales el coste del camino es irrelevante y todo lo que importa es el estado solución en sí mismo. La familia de algoritmos de búsqueda locales incluye métodos inspirados por la física estadística (**temple simulado**) y la biología evolutiva (**algoritmos genéticos**). Finalmente, la Sección 4.5 investiga la **búsqueda en línea**, en la cual el agente se enfrenta con un espacio de estados que es completamente desconocido.

4.1 Estrategias de búsqueda informada (heurísticas)

BÚSQUEDA INFORMADA

Esta sección muestra cómo una estrategia de **búsqueda informada** (la que utiliza el conocimiento específico del problema más allá de la definición del problema en sí mismo) puede encontrar soluciones de una manera más eficiente que una estrategia no informada.

BÚSQUEDA PRIMERO EL MEJOR

FUNCIÓN DE EVALUACIÓN

A la aproximación general que consideraremos se le llamará **búsqueda primero el mejor**. La búsqueda primero el mejor es un caso particular del algoritmo general de BÚSQUEDA-ÁRBOLES o de BÚSQUEDA-GRAFOS en el cual se selecciona un nodo para la expansión basada en una **función de evaluación**, $f(n)$. Tradicionalmente, se selecciona para la expansión el nodo con la evaluación más baja, porque la evaluación mide la distancia al objetivo. La búsqueda primero el mejor puede implementarse dentro de nuestro marco general de búsqueda con una cola con prioridad, una estructura de datos que mantendrá la frontera en orden ascendente de f-valores.

El nombre de «búsqueda primero el mejor» es venerable pero inexacto. A fin de cuentas, si nosotros *realmente* pudiéramos expandir primero el mejor nodo, esto no sería una búsqueda en absoluto; sería una marcha directa al objetivo. Todo lo que podemos hacer es escoger el nodo que *parece* ser el mejor según la función de evaluación. Si la función de evaluación es exacta, entonces de verdad sería el mejor nodo; en realidad, la función de evaluación no será así, y puede dirigir la búsqueda por mal camino. No obstante, nos quedaremos con el nombre «búsqueda primero el mejor», porque «búsqueda aparentemente primero el mejor» es un poco incómodo.

FUNCIÓN HEURÍSTICA

Hay una familia entera de algoritmos de BÚSQUEDA-PRIMERO-MEJOR con funciones[1] de evaluación diferentes. Una componente clave de estos algoritmos es una **función heurística**[2], denotada $h(n)$:

$h(n)$ = coste estimado del camino más barato desde el nodo n a un nodo objetivo.

Por ejemplo, en Rumanía, podríamos estimar el coste del camino más barato desde Arad a Bucarest con la distancia en línea recta desde Arad a Bucarest.

Las funciones heurísticas son la forma más común de transmitir el conocimiento adicional del problema al algoritmo de búsqueda. Estudiaremos heurísticas con más profundidad en la Sección 4.2. Por ahora, las consideraremos funciones arbitrarias específicas del problema, con una restricción: si n es un nodo objetivo, entonces $h(n) = 0$. El resto de esta sección trata dos modos de usar la información heurística para dirigir la búsqueda.

Búsqueda voraz primero el mejor

BÚSQUEDA VORAZ PRIMERO EL MEJOR

La **búsqueda voraz primero el mejor**[3] trata de expandir el nodo más cercano al objetivo, alegando que probablemente conduzca rápidamente a una solución. Así, evalúa los nodos utilizando solamente la función heurística: $f(n) = h(n)$.

DISTANCIA EN LÍNEA RECTA

Veamos cómo trabaja para los problemas de encontrar una ruta en Rumanía utilizando la heurística **distancia en línea recta**, que llamaremos h_{DLR}. Si el objetivo es Bucarest, tendremos que conocer las distancias en línea recta a Bucarest, que se muestran en la Figura 4.1. Por ejemplo, $h_{DLR}(En(Arad)) = 366$. Notemos que los valores de h_{DLR} no pueden calcularse de la descripción de problema en sí mismo. Además, debemos tener una

[1] El Ejercicio 4.3 le pide demostrar que esta familia incluye varios algoritmos familiares no informados.
[2] Una función heurística $h(n)$ toma un *nodo* como entrada, pero depende sólo del *estado* en ese nodo.
[3] Nuestra primera edición la llamó **búsqueda avara (voraz)**; otros autores la han llamado **búsqueda primero el mejor**. Nuestro uso más general del término se sigue de Pearl (1984).

cierta cantidad de experiencia para saber que h_{DLR} está correlacionada con las distancias reales del camino y es, por lo tanto, una heurística útil.

Arad	366	**Mehadia**	241
Bucarest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figura 4.1 Valores de h_{DLR}. Distancias en línea recta a Bucarest.

La Figura 4.2 muestra el progreso de una búsqueda primero el mejor avara con h_{DLR} para encontrar un camino desde Arad a Bucarest. El primer nodo a expandir desde Arad será Sibiu, porque está más cerca de Bucarest que Zerind o que Timisoara. El siguiente nodo a expandir será Fagaras, porque es la más cercana. Fagaras en su turno genera Bucarest, que es el objetivo. Para este problema particular, la búsqueda primero el mejor avara usando h_{DLR} encuentra una solución sin expandir un nodo que no esté sobre el camino solución; de ahí, que su coste de búsqueda es mínimo. Sin embargo, no es óptimo: el camino vía Sibiu y Fagaras a Bucarest es 32 kilómetros más largo que el camino por Rimnicu Vilcea y Pitesti. Esto muestra por qué se llama algoritmo «avaro» (en cada paso trata de ponerse tan cerca del objetivo como pueda).

La mínimización de $h(n)$ es susceptible de ventajas falsas. Considere el problema de ir de Iasi a Fagaras. La heurística sugiere que Neamt sea expandido primero, porque es la más cercana a Fagaras, pero esto es un callejón sin salida. La solución es ir primero a Vaslui (un paso que en realidad está más lejano del objetivo según la heurística) y luego seguir a Urziceni, Bucarest y Fagaras. En este caso, entonces, la heurística provoca nodos innecesarios para expandir. Además, si no somos cuidadosos en descubrir estados repetidos, la solución nunca se encontrará, la búsqueda oscilará entre Neamt e Iasi.

La búsqueda voraz primero el mejor se parece a la búsqueda primero en profundidad en el modo que prefiere seguir un camino hacia el objetivo, pero volverá atrás cuando llegue a un callejón sin salida. Sufre los mismos defectos que la búsqueda primero en profundidad, no es óptima, y es incompleta (porque puede ir hacia abajo en un camino infinito y nunca volver para intentar otras posibilidades). La complejidad en tiempo y espacio, del caso peor, es $O(b^m)$, donde m es la profundidad máxima del espacio de búsqueda. Con una buena función, sin embargo, pueden reducir la complejidad considerablemente. La cantidad de la reducción depende del problema particular y de la calidad de la heurística.

Figura 4.2 Etapas en una búsqueda primero el mejor avara para Bucarest utilizando la heurística distancia en línea recta h_{DLR}. Etiquetamos lo nodos con sus h-valores.

Búsqueda A*: minimizar el costo estimado total de la solución

BÚSQUEDA A*

A la forma más ampliamente conocida de la búsqueda primero el mejor se le llama **búsqueda A*** (pronunciada «búsqueda A-estrella»). Evalúa los nodos combinando $g(n)$, el coste para alcanzar el nodo, y $h(n)$, el coste de ir al nodo objetivo:

$$f(n) = g(n) + h(n)$$

Ya que la $g(n)$ nos da el coste del camino desde el nodo inicio al nodo n, y la $h(n)$ el coste estimado del camino más barato desde n al objetivo, tenemos:

$$f(n) = \text{coste más barato estimado de la solución a través de } n.$$

Así, si tratamos de encontrar la solución más barata, es razonable intentar primero el nodo con el valor más bajo de $g(n) + h(n)$. Resulta que esta estrategia es más que razonable: con tal de que la función heurística $h(n)$ satisfaga ciertas condiciones, la búsqueda A* es tanto completa como óptima.

HEURÍSTICA ADMISIBLE

La optimalidad de A* es sencilla de analizar si se usa con la BÚSQUEDA-ÁRBOLES. En este caso, A* es óptima si $h(n)$ es una **heurística admisible**, es decir, con tal de que la $h(n)$ nunca sobrestime el coste de alcanzar el objetivo. Las heurísticas admisibles son por naturaleza optimistas, porque piensan que el coste de resolver el problema es menor que el que es en realidad. Ya que $g(n)$ es el coste exacto para alcanzar n, tenemos como consecuencia inmediata que la $f(n)$ nunca sobrestima el coste verdadero de una solución a través de n.

Un ejemplo obvio de una heurística admisible es la distancia en línea recta h_{DLR} que usamos para ir a Bucarest. La distancia en línea recta es admisible porque el camino más corto entre dos puntos cualquiera es una línea recta, entonces la línea recta no puede ser una sobrestimación. En la Figura 4.3, mostramos el progreso de un árbol de búsqueda A* para Bucarest. Los valores de g se calculan desde los costos de la Figura 3.2, y los valores de h_{DLR} son los de la Figura 4.1. Notemos en particular que Bucarest aparece primero sobre la frontera en el paso (e), pero no se selecciona para la expansión porque su coste de f (450) es más alto que el de Pitesti (417). Otro modo de decir esto consiste en que podría haber una solución por Pitesti cuyo coste es tan bajo como 417, entonces el algoritmo no se conformará con una solución que cuesta 450. De este ejemplo, podemos extraer una demostración general de que A*, *utilizando la* BÚSQUEDA-ÁRBOLES, *es óptimo si la* $h(n)$ *es admisible*. Supongamos que aparece en la frontera un nodo objetivo subóptimo G_2, y que el coste de la solución óptima es C^*. Entonces, como G_2 es subóptimo y $h(G_2) = 0$ (cierto para cualquier nodo objetivo), sabemos que

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$$

Ahora considere un nodo n de la frontera que esté sobre un camino solución óptimo, por ejemplo, Pitesti en el ejemplo del párrafo anterior. (Siempre debe de haber ese nodo si existe una solución.) Si la $h(n)$ no sobrestima el coste de completar el camino solución, entonces sabemos que

$$f(n) = g(n) + h(n) \leq C^*$$

Hemos demostrado que $f(n) \leq C^* < f(G_2)$, así que G_2 no será expandido y A* debe devolver una solución óptima.

Si utilizamos el algoritmo de BÚSQUEDA-GRAFOS de la Figura 3.19 en vez de la BÚSQUEDA-ÁRBOLES, entonces esta demostración se estropea. Soluciones subóptimas pueden devolverse porque la BÚSQUEDA-GRAFOS puede desechar el camino óptimo en un estado repetido si éste no se genera primero (*véase* el Ejercicio 4.4). Hay dos modos de arreglar este problema. La primera solución es extender la BÚSQUEDA-GRAFOS de modo que deseche el camino más caro de dos caminos cualquiera encontrando al mismo nodo (*véase* la discusión de la Sección 3.5). El cálculo complementario es complicado, pero realmente garantiza la optimalidad. La segunda solución es asegurar que el camino óptimo a cualquier estado repetido es siempre el que primero seguimos (como en el caso de la búsqueda de costo uniforme). Esta propiedad se mantiene si imponemos una nueva condición a $h(n)$,

Figura 4.3 Etapas en una búsqueda A* para Bucarest. Etiquetamos lo nodos con $f = g + h$. Los valores h son las distancias en línea recta a Bucarest tomadas de la Figura 4.1.

CONSISTENCIA

MONOTONÍA

concretamente condición de **consistencia** (también llamada **monotonía**). Una heurística $h(n)$ es consistente si, para cada nodo n y cada sucesor n' de n generado por cualquier acción a, el coste estimado de alcanzar el objetivo desde n no es mayor que el coste de alcanzar n' más el coste estimado de alcanzar el objetivo desde n':

$$h(n) \leq c(n,a,n') + h(n')$$

DESIGUALDAD TRIANGULAR

Esto es una forma de la **desigualdad triangular** general, que especifica que cada lado de un triángulo no puede ser más largo que la suma de los otros dos lados. En nuestro caso, el triángulo está formado por n, n', y el objetivo más cercano a n. Es fácil demostrar (Ejercicio 4.7) que toda heurística consistente es también admisible. La consecuencia más importante de la consistencia es la siguiente: A* utilizando la BÚSQUEDA-GRAFOS es *óptimo si la $h(n)$ es consistente*.

Aunque la consistencia sea una exigencia más estricta que la admisibilidad, uno tiene que trabajar bastante para inventar heurísticas que sean admisibles, pero no consistentes. Todas la heurísticas admisibles de las que hablamos en este capítulo también son consistentes. Consideremos, por ejemplo, h_{DLR}. Sabemos que la desigualdad triangular general se satisface cuando cada lado se mide por la distancia en línea recta, y que la distancia en línea recta entre n y n' no es mayor que $c(n, a, n')$. De ahí que, h_{DLR} es una heurística consistente.

Otra consecuencia importante de la consistencia es la siguiente: *si $h(n)$ es consistente, entonces los valores de $f(n)$, a lo largo de cualquier camino, no disminuyen*. La demostración se sigue directamente de la definición de consistencia. Supongamos que n' es un sucesor de n; entonces $g(n') = g(n) + c(n, a, n')$ para alguna a, y tenemos

$$f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n)$$

Se sigue que la secuencia de nodos expandidos por A* utilizando la BÚSQUEDA-GRAFOS están en orden no decreciente de $f(n)$. De ahí que, el primer nodo objetivo seleccionado para la expansión debe ser una solución óptima, ya que todos los nodos posteriores serán al menos tan costosos.

CURVAS DE NIVEL

El hecho de que los f-costos no disminuyan a lo largo de cualquier camino significa que podemos dibujar **curvas de nivel** en el espacio de estados, como las curvas de nivel en un mapa topográfico. La Figura 4.4 muestra un ejemplo. Dentro de la curva de nivel etiquetada con 400, todos los nodos tienen la $f(n)$ menor o igual a 400, etcétera. Entonces, debido a que A* expande el nodo de frontera de f-coste más bajo, podemos ver que A* busca hacia fuera desde el nodo inicial, añadiendo nodos en bandas concéntricas de f-coste creciente.

Con la búsqueda de coste uniforme (búsqueda A* utilizando $h(n) = 0$), las bandas serán «circulares» alrededor del estado inicial. Con heurísticas más precisas, las bandas se estirarán hacia el estado objetivo y se harán más concéntricas alrededor del camino óptimo. Si C^* es el coste del camino de solución óptimo, entonces podemos decir lo siguiente:

- A* expande todos los nodos con $f(n) < C^*$.
- A* entonces podría expandir algunos nodos directamente sobre «la curva de nivel objetivo» (donde la $f(n) = C^*$) antes de seleccionar un nodo objetivo.

Por intuición, es obvio que la primera solución encontrada debe ser óptima, porque los nodos objetivos en todas las curvas de nivel siguientes tendrán el f-coste más alto, y así

Figura 4.4 Mapa de Rumanía mostrando curvas de nivel en $f = 380$, $f = 400$ y $f = 420$, con Arad como estado inicial. Los nodos dentro de cada curva de nivel tienen f-costos menores o iguales al valor de la curva de nivel.

el g-coste más alto (porque todos los nodos de objetivo tienen $h(n) = 0$). Por intuición, es también obvio que la búsqueda A* es completa. Como añadimos las bandas de f creciente, al final debemos alcanzar una banda donde f sea igual al coste del camino a un estado objetivo[4].

Note que A* no expande ningún nodo con $f(n) > C^*$ (por ejemplo, Timisoara no se expande en la Figura 4.3 aun cuando sea un hijo de la raíz). Decimos que el subárbol debajo de Timisoara está **podado**; como h_{DLR} es admisible, el algoritmo seguramente no hará caso de este subárbol mientras todavía se garantiza la optimalidad. El concepto de poda (eliminación de posibilidades a considerar sin necesidad de examinarlas) es importante para muchas áreas de IA.

PODA

Una observación final es que entre los algoritmos óptimos de este tipo (los algoritmos que extienden los caminos de búsqueda desde la raíz) A* es **óptimamente eficiente** para cualquier función heurística. Es decir, ningún otro algoritmo óptimo garantiza expandir menos nodos que A* (excepto posiblemente por los desempates entre los nodos con $f(n) = C^*$). Esto es porque cualquier algoritmo que no expanda todos los nodos con $f(n) < C^*$ corre el riesgo de omitir la solución óptima.

ÓPTIMAMENTE EFICIENTE

Nos satisface bastante la búsqueda A* ya que es completa, óptima, y óptimamente eficiente entre todos los algoritmos. Lamentablemente, no significa que A* sea la respuesta a todas nuestras necesidades de búsqueda. La dificultad es que, para la mayoría de los problemas, el número de nodos dentro de la curva de nivel del objetivo en el espacio de búsqueda es todavía exponencial en la longitud de la solución. Aunque la demostración del resultado está fuera del alcance de este libro, se ha mostrado que el

[4] La completitud requiere que haya sólo un número finito de nodos con el coste menor o igual a C *, una condición que es cierta si todos los costos exceden algún número ϵ finito y si b es finito.

crecimiento exponencial ocurrirá a no ser que el error en la función heurística no crezca más rápido que el logaritmo del coste de camino real. En notación matemática, la condición para el crecimiento subexponencial es

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

donde $h^*(n)$ es el coste real de alcanzar el objetivo desde n. En la práctica, para casi todas las heurísticas, el error es al menos proporcional al coste del camino, y el crecimiento exponencial que resulta alcanza la capacidad de cualquier computador. Por esta razón, es a menudo poco práctico insistir en encontrar una solución óptima. Uno puede usar las variantes de A* que encuentran rápidamente soluciones subóptimas, o uno a veces puede diseñar heurísticas que sean más exactas, pero no estrictamente admisibles. En cualquier caso, el empleo de buenas heurísticas proporciona enormes ahorros comparados con el empleo de una búsqueda no informada. En la Sección 4.2, veremos el diseño de heurísticas buenas.

El tiempo computacional no es, sin embargo, la desventaja principal de A*. Como mantiene todos los nodos generados en memoria (como hacen todos los algoritmos de BÚSQUEDA-GRAFOS), A*, por lo general, se queda sin mucho espacio antes de que se quede sin tiempo. Por esta razón, A* no es práctico para problemas grandes. Los algoritmos recientemente desarrollados han vencido el problema de espacio sin sacrificar la optimalidad o la completitud, con un pequeño coste en el tiempo de ejecución. Éstos se discutirán a continuación.

Búsqueda heurística con memoria acotada

La forma más simple de reducir la exigencia de memoria para A* es adaptar la idea de profundizar iterativamente al contexto de búsqueda heurística, resultando así el algoritmo A* de profundidad iterativa (A*PI). La diferencia principal entre A*PI y la profundidad iterativa estándar es que el corte utilizado es el f-coste ($g + h$) más que la profundidad; en cada iteración, el valor del corte es el f-coste más pequeño de cualquier nodo que excedió el corte de la iteración anterior. A*PI es práctico para muchos problemas con costos unidad y evita el trabajo asociado con el mantenimiento de una cola ordenada de nodos. Lamentablemente, esto sufre de las mismas dificultades con costos de valores reales como hace la versión iterativa de búsqueda de coste uniforme descrita en el Ejercicio 3.11. Esta sección brevemente examina dos algoritmos más recientes con memoria acotada, llamados BRPM y A*M.

BÚSQUEDA RECURSIVA DEL PRIMERO MEJOR

La **búsqueda recursiva del primero mejor** (BRPM) es un algoritmo sencillo recursivo que intenta imitar la operación de la búsqueda primero el mejor estándar, pero utilizando sólo un espacio lineal. En la Figura 4.5 se muestra el algoritmo. Su estructura es similar a la búsqueda primero en profundidad recursiva, pero más que seguir indefinidamente hacia abajo el camino actual, mantiene la pista del f-valor del mejor camino alternativo disponible desde cualquier antepasado del nodo actual. Si el nodo actual excede este límite, la recursividad vuelve atrás al camino alternativo. Como la recursividad vuelve atrás, la BRPM sustituye los f-valores de cada nodo a lo largo del camino con el mejor f-valor de su hijo. De este modo, la BRPM recuerda el f-valor de la mejor hoja en el subárbol olvidado y por lo tanto puede decidir si merece la pena expandir el subárbol más tarde. La Figura 4.6 muestra cómo la BRPM alcanza Bucarest.

```
función BÚSQUEDA-RECURSIVA-PRIMERO-MEJOR(problema) devuelve una solución, o fallo
  BRPM(problema, HACER-NODO(ESTADO-INICIAL[problema]),∞)

función BRPM(problema,nodo,f_límite) devuelve una solución, o fallo y un nuevo límite
f-costo
  si TEST-OBJETIVO[problema](estado) entonces devolver nodo
  sucesores ← EXPANDIR(nodo,problema)
  si sucesores está vacío entonces devolver fallo, ∞
  para cada s en sucesores hacer
    f[s] ← max(g(s) + h(s), f[nodo])
  repetir
    mejor ← nodo con f-valor más pequeño de sucesores
    si f[mejor] > f_límite entonces devolver fallo, f[mejor]
    alternativa ← nodo con el segundo f-valor más pequeño entre los sucesores
    resultado,f[mejor] ← BRPM(problema,mejor,min(f_límite,alternativa))
    si resultado ≠ fallo entonces devolver resultado
```

Figura 4.5 Algoritmo para la búsqueda primero el mejor recursiva.

La BRPM es algo más eficiente que A*PI, pero todavía sufre de la regeneración excesiva de nodos. En el ejemplo de la Figura 4.6, la BRPM sigue primero el camino vía Rimnicu Vilcea, entonces «cambia su opinión» e intenta Fagaras, y luego cambia su opinión hacia atrás otra vez. Estos cambios de opinión ocurren porque cada vez que el mejor camino actual se extiende, hay una buena posibilidad que aumente su *f*-valor (*h* es por lo general menos optimista para nodos más cercanos al objetivo). Cuando esto pasa, en particular en espacios de búsqueda grandes, el segundo mejor camino podría convertirse en el mejor camino, entonces la búsqueda tiene que retroceder para seguirlo. Cada cambio de opinión corresponde a una iteración de A*PI, y podría requerir muchas nuevas expansiones de nodos olvidados para volver a crear el mejor camino y ampliarlo en un nodo más.

Como A*, BRPM es un algoritmo óptimo si la función heurística $h(n)$ es admisible. Su complejidad en espacio es $O(bd)$, pero su complejidad en tiempo es bastante difícil de caracterizar: depende de la exactitud de la función heurística y de cómo cambia a menudo el mejor camino mientras se expanden los nodos. Tanto A*PI como BRPM están sujetos al aumento potencialmente exponencial de la complejidad asociada con la búsqueda en grafos (*véase* la Sección 3.5), porque no pueden comprobar para saber si hay estados repetidos con excepción de los que están en el camino actual. Así, pueden explorar el mismo estado muchas veces.

A*PI y BRPM sufren de utilizar *muy poca* memoria. Entre iteraciones, A*PI conserva sólo un número: el límite *f*-coste actual. BRPM conserva más información en la memoria, pero usa sólo $O(bd)$ de memoria: incluso si hubiera más memoria disponible, BRPM no tiene ningún modo de aprovecharse de ello.

A*M

A*MS

Parece sensible, por lo tanto, usar toda la memoria disponible. Dos algoritmos que hacen esto son **A*M** (A* con memoria acotada) y **A*MS** (A*M simplificada). Describiremos A*MS, que es más sencillo. A*MS avanza como A*, expandiendo la mejor hoja

Figura 4.6 Etapas en una búsqueda BRPM para la ruta más corta a Bucarest. Se muestra el valor del *f*-límite para cada llamada recursiva sobre cada nodo actual. (a) Se sigue el camino vía Rimnicu Vilcea hasta que la mejor hoja actual (Pitesti) tenga un valor que es peor que el mejor camino alternativo (Fagaras). (b) La recursividad se aplica y el mejor valor de las hojas del subárbol olvidado (417) se le devuelve hacia atrás a Rimnicu Vilcea; entonces se expande Fagaras, revela un mejor valor de hoja de 450. (c) La recursividad se aplica y el mejor valor de las hojas del subárbol olvidado (450) se le devuelve hacia atrás a Fagaras; entonces se expande Rimnicu Vilcea. Esta vez, debido a que el mejor camino alternativo (por Timisoara) cuesta por lo menos 447, la expansión sigue por Bucarest.

hasta que la memoria esté llena. En este punto, no se puede añadir un nuevo nodo al árbol de búsqueda sin retirar uno viejo. A*MS siempre retira el peor nodo hoja (el de *f*-valor más alto). Como en la BRPM, A*MS entonces devuelve hacia atrás, a su padre, el valor del nodo olvidado. De este modo, el antepasado de un subárbol olvidado sabe la ca-

lidad del mejor camino en el subárbol. Con esta información, A*MS vuelve a generar el subárbol sólo cuando *todos los otros caminos* parecen peores que el camino olvidado. Otro modo de decir esto consiste en que, si todos los descendientes de un nodo *n* son olvidados, entonces no sabremos por qué camino ir desde *n*, pero todavía tendremos una idea de cuánto vale la pena ir desde *n* a cualquier nodo.

El algoritmo completo es demasiado complicado para reproducirse aquí[5], pero hay un matiz digno de mencionar. Dijimos que A*MS expande la mejor hoja y suprime la peor hoja. ¿Y si *todos* los nodos hoja tienen el mismo *f*-valor? Entonces el algoritmo podría seleccionar el mismo nodo para eliminar y expandir. A*MS soluciona este problema expandiendo la mejor hoja *más nueva* y suprimiendo la peor hoja *más vieja*. Estos pueden ser el mismo nodo sólo si hay una sola hoja; en ese caso, el árbol actual de búsqueda debe ser un camino sólo desde la raíz a la hoja llenando toda la memoria. Si la hoja no es un nodo objetivo, entonces, *incluso si está sobre un camino solución óptimo*, esa solución no es accesible con la memoria disponible. Por lo tanto, el nodo puede descartarse como si no tuviera ningún sucesor.

A*MS es completo si hay alguna solución alcanzable, es decir, si *d*, la profundidad del nodo objetivo más superficial, es menor que el tamaño de memoria (expresada en nodos). Es óptimo si cualquier solución óptima es alcanzable; de otra manera devuelve la mejor solución alcanzable. En términos prácticos, A*MS bien podría ser el mejor algoritmo de uso general para encontrar soluciones óptimas, en particular cuando el espacio de estados es un grafo, los costos no son uniformes, y la generación de un nodo es costosa comparada con el gasto adicional de mantener las listas abiertas y cerradas.

Sobre problemas muy difíciles, sin embargo, a menudo al A*MS se le fuerza a cambiar hacia delante y hacia atrás continuamente entre un conjunto de caminos solución candidatos, y sólo un pequeño subconjunto de ellos puede caber en memoria (esto se parece al problema de *thrashing* en sistemas de paginación de disco). Entonces el tiempo extra requerido para la regeneración repetida de los mismos nodos significa que los problemas que serían prácticamente resolubles por A*, considerando la memoria ilimitada, se harían intratables para A*MS. Es decir, *las limitaciones de memoria pueden hacer a un problema intratable desde el punto de vista de tiempo de cálculo*. Aunque no haya ninguna teoría para explicar la compensación entre el tiempo y la memoria, parece que esto es un problema ineludible. La única salida es suprimir la exigencia de optimización.

THRASHING

Aprender a buscar mejor

Hemos presentado varias estrategias fijas (primero en anchura, primero el mejor avara, etcétera) diseñadas por informáticos. ¿Podría un agente aprender a buscar mejor? La respuesta es sí, y el método se apoya sobre un concepto importante llamado el **espacio de estados metanivel**. Cada estado en un espacio de estados metanivel captura el estado interno (computacional) de un programa que busca en un **espacio de estados a nivel de objeto** como Rumanía. Por ejemplo, el estado interno del algoritmo A* consiste en el

ESPACIO DE ESTADOS METANIVEL

ESPACIO DE ESTADOS A NIVEL DE OBJETO

[5] Un esbozo apareció en la primera edición de este libro.

árbol actual de búsqueda. Cada acción en el espacio de estados metanivel es un paso de cómputo que cambia el estado interno; por ejemplo, cada paso de cómputo en A* expande un nodo hoja y añade sus sucesores al árbol. Así, la Figura 4.3, la cual muestra una secuencia de árboles de búsqueda más y más grandes, puede verse como la representación de un camino en el espacio de estados metanivel donde cada estado sobre el camino es un árbol de búsqueda a nivel de objeto.

APRENDIZAJE METANIVEL

Ahora, el camino en la Figura 4.3 tiene cinco pasos, incluyendo un paso, la expansión de Fagaras, que no es especialmente provechoso. Para problemas más difíciles, habrá muchos de estos errores, y un algoritmo de **aprendizaje metanivel** puede aprender de estas experiencias para evitar explorar subárboles no prometedores. Las técnicas usadas para esta clase de aprendizaje están descritas en el Capítulo 21. El objetivo del aprendizaje es reducir al mínimo el **coste total** de resolver el problema, compensar el costo computacional y el coste del camino.

4.2 Funciones heurísticas

En esta sección, veremos heurísticas para el 8-puzle, para que nos den información sobre la naturaleza de las heurísticas en general.

El 8-puzle fue uno de los primeros problemas de búsqueda heurística. Como se mencionó en la Sección 3.2, el objeto del puzle es deslizar las fichas horizontalmente o verticalmente al espacio vacío hasta que la configuración empareje con la configuración objetivo (Figura 4.7).

Figura 4.7 Un caso típico del 8-puzle. La solución tiene 26 pasos.

El coste medio de la solución para los casos generados al azar del 8-puzle son aproximadamente 22 pasos. El factor de ramificación es aproximadamente tres. (Cuando la ficha vacía está en el medio, hay cuatro movimientos posibles; cuando está en una esquina hay dos; y cuando está a lo largo de un borde hay tres.) Esto significa que una búsqueda exhaustiva a profundidad 22 miraría sobre $3^{22} \approx 3{,}1 \times 10^{10}$ estados. Manteniendo la pista de los estados repetidos, podríamos reducirlo a un factor de aproximadamente 170.000, porque hay sólo $9!/2 = 181.440$ estados distintos que son alcanzables.

(*Véase* el Ejercicio 3.4.) Esto es un número manejable, pero el número correspondiente para el puzle-15 es aproximadamente 10^{13}, entonces lo siguiente es encontrar una buena función heurística. Si queremos encontrar las soluciones más cortas utilizando A*, necesitamos una función heurística que nunca sobrestima el número de pasos al objetivo. Hay una larga historia de tales heurísticas para el 15-puzle; aquí están dos candidatas comúnmente usadas:

- h_1 = número de piezas mal colocadas. Para la Figura 4.7, las 8 piezas están fuera de su posición, así que el estado inicial tiene $h_1 = 8$. h_1 es una heurística admisible, porque está claro que cualquier pieza que está fuera de su lugar debe moverse por lo menos una vez.
- h_2 = suma de las distancias de las piezas a sus posiciones en el objetivo. Como las piezas no pueden moverse en diagonal, la distancia que contaremos será la suma de las distancias horizontales y verticales. Esto se llama a veces la **distancia en la ciudad** o **distancia de Manhattan**. h_2 es también admisible, porque cualquier movimiento que se puede hacer es mover una pieza un paso más cerca del objetivo. Las piezas 1 a 8 en el estado inicial nos dan una distancia de Manhattan de

DISTANCIA DE MANHATTAN

$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

Como era de esperar, ninguna sobrestima el coste solución verdadero, que es 26.

El efecto de la precisión heurística en el rendimiento

FACTOR DE RAMIFICACIÓN EFICAZ

Una manera de caracterizar la calidad de una heurística es el b^* **factor de ramificación eficaz**. Si el número total de nodos generados por A* para un problema particular es N, y la profundidad de la solución es d, entonces b^* es el factor de ramificación que un árbol uniforme de profundidad d debería tener para contener $N + 1$ nodos. Así,

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

Por ejemplo, si A* encuentra una solución a profundidad cinco utilizando 52 nodos, entonces el factor de ramificación eficaz es 1,92. El factor de ramificación eficaz puede variar según los ejemplos del problema, pero por lo general es constante para problemas suficientemente difíciles. Por lo tanto, las medidas experimentales de b^* sobre un pequeño conjunto de problemas pueden proporcionar una buena guía para la utilidad total de la heurística. Una heurística bien diseñada tendría un valor de b^* cerca de 1, permitiría resolver problemas bastante grandes.

Para probar las funciones heurísticas h_1 y h_2, generamos 1.200 problemas aleatorios con soluciones de longitudes de 2 a 24 (100 para cada número par) y los resolvemos con la búsqueda de profundidad iterativa y con la búsqueda en árbol A* usando tanto h_1 como h_2. La Figura 4.8 nos da el número medio de nodos expandidos por cada estrategia y el factor de ramificación eficaz. Los resultados sugieren que h_2 es mejor que h_1, y es mucho mejor que la utilización de la búsqueda de profundidad iterativa. Sobre nuestras soluciones con longitud 14, A* con h_2 es 30.000 veces más eficiente que la búsqueda no informada de profundidad iterativa.

Uno podría preguntarse si h_2 es siempre mejor que h_1. La respuesta es sí. Es fácil ver de las definiciones de las dos heurísticas que, para cualquier nodo n, $h_2(n) \geq h_1(n)$.

DOMINACIÓN

Así decimos que h_2 **domina** a h_1. La dominación se traslada directamente a la eficiencia: A* usando h_2 nunca expandirá más nodos que A* usando h_1 (excepto posiblemente para algunos nodos con $f(n) = C^*$). El argumento es simple. Recuerde la observación de la página 113 de que cada nodo con $f(n) < C^*$ será seguramente expandido. Esto es lo mismo que decir que cada nodo con $h(n) < C^* - g(n)$ será seguramente expandido. Pero debido a que h_2 es al menos tan grande como h_1 para todos los nodos, cada nodo que seguramente será expandido por la búsqueda A* con h_2 será seguramente también expandido con h_1, y h_1 podría también hacer que otros nodos fueran expandidos. De ahí que es siempre mejor usar una función heurística con valores más altos, a condición de que no sobrestime y que el tiempo computacional de la heurística no sea demasiado grande.

	Costo de la búsqueda			Factor de ramificación eficaz		
d	BPI	A*(h_1)	A*(h_2)	BPI	A*(h_1)	A*(h_2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	—	539	113	—	1,44	1,23
16	—	1301	211	—	1,45	1,25
18	—	3056	363	—	1,46	1,26
20	—	7276	676	—	1,47	1,27
22	—	18094	1219	—	1,48	1,28
24	—	39135	1641	—	1,48	1,26

Figura 4.8 Comparación de los costos de la búsqueda y factores de ramificación eficaces para la BÚSQUEDA-PROFUNDIDAD-ITERATIVA y los algoritmos A* con h_1 y h_2. Los datos son la media de 100 ejemplos del puzle-8, para soluciones de varias longitudes.

Inventar funciones heurísticas admisibles

Hemos visto que h_1 (piezas más colocadas) y h_2 (distancia de Manhattan) son heurísticas bastante buenas para el 8-puzle y que h_2 es mejor. ¿Cómo ha podido surgir h_2? ¿Es posible para un computador inventar mecánicamente tal heurística?

h_1, h_2 son estimaciones de la longitud del camino restante para el 8-puzle, pero también son longitudes de caminos absolutamente exactos para versiones *simplificadas* del puzle. Si se cambiaran las reglas del puzle de modo que una ficha pudiera moverse a todas partes, en vez de solamente al cuadrado adyacente vacío, entonces h_1 daría el número exacto de pasos en la solución más corta. Del mismo modo, si una ficha pudiera moverse a un cuadrado en cualquier dirección, hasta en un cuadrado ocupado, entonces h_2 daría el número exacto de pasos en la solución más corta. A un problema con menos restricciones en las acciones se le llama **problema relajado**. *El costo de una solución óptima en un problema relajado es una heurística admisible para el problema original.*

PROBLEMA RELAJADO

La heurística es admisible porque la solución óptima en el problema original es, por definición, también una solución en el problema relajado y por lo tanto debe ser al menos tan cara como la solución óptima en el problema relajado. Como la heurística obtenida es un costo exacto para el problema relajado, debe cumplir la desigualdad triangular y es por lo tanto **consistente** (*véase* la página 113).

Si la definición de un problema está escrita en un lenguaje formal, es posible construir problemas relajados automáticamente[6]. Por ejemplo, si las acciones del 8-puzle están descritas como

Una ficha puede moverse del cuadrado A al cuadrado B si
A es horizontalmente o verticalmente adyacente a B **y** B es la vacía

podemos generar tres problemas relajados quitando una o ambas condiciones:

(a) Una ficha puede moverse del cuadrado A al cuadrado B si A es adyacente a B.
(b) Una ficha puede moverse del cuadrado A al cuadrado B si B es el vacío.
(c) Una ficha puede moverse del cuadrado A al cuadrado B.

De (a), podemos obtener h_2 (distancia de Manhattan). El razonamiento es que h_2 sería el resultado apropiado si moviéramos cada ficha en dirección a su destino. La heurística obtenida de (b) se discute en el Ejercicio 4.9. De (c), podemos obtener h_1 (fichas mal colocadas), porque sería el resultado apropiado si las fichas pudieran moverse a su destino en un paso. Notemos que es crucial que los problemas relajados generados por esta técnica puedan resolverse esencialmente sin búsqueda, porque las reglas relajadas permiten que el problema sea descompuesto en ocho subproblemas independientes. Si el problema relajado es difícil de resolver, entonces los valores de la correspondencia heurística serán costosos de obtener[7].

Un programa llamado ABSOLVER puede generar heurísticas automáticamente a partir de las definiciones del problema, usando el método del «problema relajado» y otras técnicas (Prieditis, 1993). ABSOLVER generó una nueva heurística para el 8-puzle mejor que cualquier heurística y encontró el primer heurístico útil para el famoso puzle cubo de Rubik.

Un problema con la generación de nuevas funciones heurísticas es que a menudo se falla al conseguir una heurística «claramente mejor». Si tenemos disponible un conjunto de heurísticas admisibles $h_1 \ldots h_m$ para un problema, y ninguna de ellas domina a las demás, ¿qué deberíamos elegir? No tenemos por qué hacer una opción. Podemos tener lo mejor de todas, definiendo

$$h(n) = \max \{h_1(n), \ldots, h_m(n)\}$$

Esta heurística compuesta usando cualquier función es más exacta sobre el nodo en cuestión. Como las heurísticas componentes son admisibles, h es admisible; es tam-

[6] En los capítulos 8 y 11, describiremos lenguajes formales convenientes para esta tarea; con descripciones formales que puedan manipularse, puede automatizarse la construcción de problemas relajados. Por el momento, usaremos el castellano.

[7] Note que una heurística perfecta puede obtenerse simplemente permitiendo a h ejecutar una búsqueda primero en anchura «a escondidas». Así, hay una compensación entre exactitud y tiempo de cálculo para las funciones heurísticas.

bién fácil demostrar que h es consistente. Además, h domina a todas sus heurísticas componentes.

SUBPROBLEMA

También se pueden obtener heurísticas admisibles del coste de la solución de un **subproblema** de un problema dado. Por ejemplo, la Figura 4.9 muestra un subproblema del puzle-8 de la Figura 4.7. El subproblema implica la colocación de las fichas 1, 2, 3, 4 en sus posiciones correctas. Claramente, el coste de la solución óptima de este subproblema es una cota inferior sobre el coste del problema completo. Parece ser considerablemente más exacta que la distancia de Manhattan, en algunos casos.

MODELO DE BASES DE DATOS

La idea que hay detrás del **modelo de bases de datos** es almacenar estos costos exactos de las soluciones para cada posible subproblema (en nuestro ejemplo, cada configuración posible de las cuatro fichas y el vacío; note que las posiciones de las otras cuatro fichas son irrelevantes para los objetivos de resolver el subproblema, pero los movimientos de esas fichas cuentan realmente hacia el coste). Entonces, calculamos una heurística admisible h_{BD}, para cada estado completo encontrado durante una búsqueda, simplemente mirando la configuración del subproblema correspondiente en la base de datos. La base de datos se construye buscando hacia atrás desde el estado objetivo y registrando el coste de cada nuevo modelo encontrado; el gasto de esta búsqueda se amortiza sobre los siguientes problemas.

La opción de 1-2-3-4 es bastante arbitraria; podríamos construir también bases de datos para 5-6-7-8, y para 2-4-6-8, etcétera. Cada base de datos produce una heurística admisible, y esta heurística puede combinarse, como se explicó antes, tomando el valor máximo. Una heurística combinada de esta clase es mucho más exacta que la distancia de Manhattan; el número de nodos generados, resolviendo 15-puzles aleatorios, puede reducirse en un factor de 1.000.

Uno podría preguntarse si las heurísticas obtenidas de las bases de datos 1-2-3-4 y 5-6-7-8 podrían sumarse, ya que los dos subproblemas parecen no superponerse. ¿Esto daría aún una heurística admisible? La respuesta es no, porque las soluciones del subproblema 1-2-3-4 y del subproblema 5-6-7-8 para un estado compartirán casi seguramente algunos movimientos (es improbable que 1-2-3-4 pueda colocarse en su lugar sin tocar 5-6-7-8, y *viceversa*). ¿Pero y si no contamos estos movimientos? Es decir no registramos el costo total para resolver el problema 1-2-3-4, sino solamente el número de mo-

Figura 4.9 Un subproblema del puzle-8 dado en la Figura 4.7. La tarea es conseguir las fichas 1, 2, 3 y 4 en sus posiciones correctas, sin preocuparse de lo que le pasan a las otras fichas.

vimientos que implican 1-2-3-4. Entonces es fácil ver que la suma de los dos costos todavía es una cota inferior del costo de resolver el problema entero. Esta es la idea que hay detrás del **modelo de bases de datos disjuntas**. Usando tales bases de datos, es posible resolver puzles-15 aleatorios en milisegundos (el número de nodos generados se reduce en un factor de 10.000 comparado con la utilización de la distancia de Manhattan). Para puzles-24, se puede obtener una aceleración de aproximadamente un millón.

MODELO DE BASES DE DATOS DISJUNTAS

El modelo de bases de datos disjuntas trabajan para puzles de deslizamiento de fichas porque el problema puede dividirse de tal modo que cada movimiento afecta sólo a un subproblema, ya que sólo se mueve una ficha a la vez. Para un problema como el cubo de Rubik, esta clase de subdivisión no puede hacerse porque cada movimiento afecta a ocho o nueve de los 25 cubos. Actualmente, no está claro cómo definir bases de datos disjuntas para tales problemas.

Aprendizaje de heurísticas desde la experiencia

Una función heurística $h(n)$, como se supone, estima el costo de una solución que comienza desde el estado en el nodo n. ¿Cómo podría un agente construir tal función? Se dio una solución en la sección anterior (idear problemas relajados para los cuales puede encontrarse fácilmente una solución óptima). Otra solución es aprender de la experiencia. «La experiencia» aquí significa la solución de muchos 8-puzles, por ejemplo. Cada solución óptima en un problema del 8-puzle proporciona ejemplos para que pueda aprender la función $h(n)$. Cada ejemplo se compone de un estado del camino solución y el costo real de la solución desde ese punto. A partir de estos ejemplos, se puede utilizar un algoritmo de **aprendizaje inductivo** para construir una función $h(n)$ que pueda (con suerte) predecir los costos solución para otros estados que surjan durante la búsqueda. Las técnicas para hacer esto utilizando redes neuronales, árboles de decisión, y otros métodos, se muestran en el Capítulo 18 (los métodos de aprendizaje por refuerzo, también aplicables, serán descritos en el Capítulo 21).

CARACTERÍSTICAS

Los métodos de aprendizaje inductivos trabajan mejor cuando se les suministran **características** de un estado que sean relevantes para su evaluación, más que sólo la descripción del estado. Por ejemplo, la característica «número de fichas mal colocadas» podría ser útil en la predicción de la distancia actual de un estado desde el objetivo. Llamemos a esta característica $x_1(n)$. Podríamos tomar 100 configuraciones del 8-puzle generadas aleatoriamente y unir las estadísticas de sus costos de la solución actual. Podríamos encontrar que cuando $x_1(n)$ es cinco, el coste medio de la solución está alrededor de 14, etcétera. Considerando estos datos, el valor de x_1 puede usarse para predecir $h(n)$. Desde luego, podemos usar varias características. Una segunda característica $x_2(n)$ podría ser «el número de pares de fichas adyacentes que son también adyacentes en el estado objetivo». ¿Cómo deberían combinarse $x_1(n)$ y $x_2(n)$ para predecir $h(n)$? Una aproximación común es usar una combinación lineal:

$$h(n) = c_1 x_1(n) + c_2 x_2(n)$$

Las constantes c_1 y c_2 se ajustan para dar el mejor ajuste a los datos reales sobre los costos de la solución. Presumiblemente, c_1 debe ser positivo y c_2 debe ser negativo.

4.3 Algoritmos de búsqueda local y problemas de optimización

Los algoritmos de búsqueda que hemos visto hasta ahora se diseñan para explorar sistemáticamente espacios de búsqueda. Esta forma sistemática se alcanza manteniendo uno o más caminos en memoria y registrando qué alternativas se han explorado en cada punto a lo largo del camino y cuáles no. Cuando se encuentra un objetivo, el *camino* a ese objetivo también constituye una *solución* al problema.

En muchos problemas, sin embargo, el camino al objetivo es irrelevante. Por ejemplo, en el problema de las 8-reinas (*véase* la página 74), lo que importa es la configuración final de las reinas, no el orden en las cuales se añaden. Esta clase de problemas incluyen muchas aplicaciones importantes como diseño de circuitos integrados, disposición del suelo de una fábrica, programación del trabajo en tiendas, programación automática, optimización de redes de telecomunicaciones, dirigir un vehículo, y la gestión de carteras.

BÚSQUEDA LOCAL

ESTADO ACTUAL

Si no importa el camino al objetivo, podemos considerar una clase diferente de algoritmos que no se preocupen en absoluto de los caminos. Los algoritmos de **búsqueda local** funcionan con un solo **estado actual** (más que múltiples caminos) y generalmente se mueve sólo a los vecinos del estado. Típicamente, los caminos seguidos por la búsqueda no se retienen. Aunque los algoritmos de búsqueda local no son sistemáticos, tienen dos ventajas claves: (1) usan muy poca memoria (por lo general una cantidad constante); y (2) pueden encontrar a menudo soluciones razonables en espacios de estados grandes o infinitos (continuos) para los cuales son inadecuados los algoritmos sistemáticos.

Figura 4.10 Un paisaje del espacio de estados unidimensional en el cual la elevación corresponde a la función objetivo. El objetivo es encontrar el máximo global. La búsqueda de ascensión de colinas modifica el estado actual para tratar de mejorarlo, como se muestra con la flecha. Se definen en el texto varios rasgos topográficos.

PROBLEMAS DE OPTIMIZACIÓN

FUNCIÓN OBJETIVO

Además de encontrar los objetivos, los algoritmos de búsqueda local son útiles para resolver **problemas de optimización** puros, en los cuales el objetivo es encontrar el mejor estado según una **función objetivo**. Muchos problemas de optimización no encajan en el modelo «estándar» de búsqueda introducido en el Capítulo 3. Por ejemplo, la naturaleza proporciona una función objetivo (idoneidad o salud reproductiva) que la evolución Darwiniana intenta optimizar, pero no hay ningún «test objetivo» y ningún «coste de camino» para este problema.

PAISAJE DEL ESPACIO DE ESTADOS

MÍNIMO GLOBAL

MÁXIMO GLOBAL

Para entender la búsqueda local, encontraremos muy útil considerar la forma o el **paisaje del espacio de estados** (como en la Figura 4.10). El paisaje tiene «posición» (definido por el estado) y «elevación» (definido por el valor de la función de coste heurística o función objetivo). Si la elevación corresponde al costo, entonces el objetivo es encontrar el valle más bajo (un **mínimo global**); si la elevación corresponde a una función objetivo, entonces el objetivo es encontrar el pico más alto (un **máximo global**). (Puedes convertir uno en el otro solamente al insertar un signo menos.) Los algoritmos de búsqueda local exploran este paisaje. Un algoritmo de búsqueda local completo siempre encuentra un objetivo si existe; un algoritmo óptimo siempre encuentran un mínimo/máximo global.

Búsqueda de ascensión de colinas

ASCENSIÓN DE COLINAS

En la Figura 4.11 se muestra el algoritmo de búsqueda de **ascensión de colinas**. Es simplemente un bucle que continuamente se mueve en dirección del valor creciente, es decir, cuesta arriba. Termina cuando alcanza «un pico» en donde ningún vecino tiene un valor más alto. El algoritmo no mantiene un árbol de búsqueda, sino una estructura de datos del nodo actual que necesita sólo el registro del estado y su valor de función objetivo. La ascensión de colinas no mira delante más allá de los vecinos inmediatos del estado actual. Se parece a lo que hacemos cuando tratamos de encontrar la cumbre del Everest en una niebla mientras sufrimos amnesia.

Para ilustrar la ascensión de colinas, usaremos **el problema de las 8-reinas** introducido en la página 74. Los algoritmos de búsqueda local típicamente usan una **formula-**

```
función ASCENSIÓN-COLINAS(problema) devuelve un estado que es un máximo local
  entradas: problema, un problema
  variables locales: actual, un nodo
                     vecino, un nodo

  actual ← HACER-NODO(ESTADO-INICIAL[problema])
  bucle hacer
    vecino ← sucesor de valor más alto de actual
    si VALOR[vecino] ≤ VALOR[actual] entonces devolver ESTADO[actual]
    actual ← vecino
```

Figura 4.11 El algoritmo de búsqueda ascensión de colinas (la versión de subida más rápida), que es la técnica de búsqueda local más básica. En cada paso el nodo actual se sustituye por el mejor vecino; en esta versión, el vecino con el VALOR más alto, pero si se utiliza una heurística *h* de estimación de costos, sería el vecino con *h* más bajo.

ción de estados completa, donde cada estado tiene a ocho reinas sobre el tablero, una por columna. La función sucesor devuelve todos los estados posibles generados moviendo una reina a otro cuadrado en la misma columna (entonces cada estado tiene $8 \times 7 = 56$ sucesores). La función de costo heurística h es el número de pares de reinas que se atacan la una a la otra, directa o indirectamente. El mínimo global de esta función es cero, que ocurre sólo en soluciones perfectas. La Figura 4.12(a) muestra un estado con $h = 17$. La figura también muestra los valores de todos sus sucesores, con los mejores sucesores que tienen $h = 12$. Los algoritmos de ascensión de colinas eligen típicamente al azar entre el conjunto de los mejores sucesores, si hay más de uno.

Figura 4.12 (a) Un estado de 8-reinas con una heurística de estimación de costos $h = 17$, mostrando al valor de h para cada sucesor posible obtenido al mover una reina dentro de su columna. Los mejores movimientos están marcados. (b) Un mínimo local en el espacio de estados de las 8-reinas; el estado tiene $h = 1$ pero cada sucesor tiene un coste más alto.

BÚSQUEDA LOCAL VORAZ

A veces a la ascensión de colinas se le llama **búsqueda local voraz** porque toma un estado vecino bueno sin pensar hacia dónde ir después. Aunque la avaricia sea considerada uno de los siete pecados mortales, resulta que los algoritmos avaros a menudo funcionan bastante bien. La ascensión de colinas a menudo hace el progreso muy rápido hacia una solución, porque es por lo general bastante fácil mejorar un estado malo. Por ejemplo, desde el estado de la Figura 4.12(a), se realizan solamente cinco pasos para alcanzar el estado de la Figura 4.12(b), que tiene $h = 1$ y es casi una solución. Lamentablemente, la ascensión de colinas a menudo se atasca por los motivos siguientes:

- **Máximo local:** un máximo local es un pico que es más alto que cada uno de sus estados vecinos, pero más abajo que el máximo global. Los algoritmos de ascensión de colinas que alcanzan la vecindad de un máximo local irán hacia el pico, pero entonces se atascarán y no podrán ir a ninguna otra parte. La Figura 4.10 ilustra el problema esquemáticamente. Más concretamente, el estado en la Figura 4.12(b) es

de hecho un máximo local (es decir, un mínimo local para el coste h); cada movimiento de una sola reina hace la situación peor.

- **Crestas**: la Figura 4.13 muestra una cresta. Las crestas causan una secuencia de máximos locales que hace muy difícil la navegación para los algoritmos avaros.
- **Meseta**: una meseta es un área del paisaje del espacio de estados donde la función de evaluación es plana. Puede ser un máximo local plano, del que no existe ninguna salida ascendente, o una **terraza**, por la que se pueda avanzar (*véase* la Figura 4.10). Una búsqueda de ascensión de colinas podría ser incapaz de encontrar su camino en la meseta.

TERRAZA

En cada caso, el algoritmo alcanza un punto en el cual no se puede hacer ningún progreso. Comenzando desde un estado de las ocho reinas generado aleatoriamente, la ascensión de colinas por la zona más escarpada se estanca el 86 por ciento de las veces, y resuelve sólo el 14 por ciento de los problemas. Trabaja rápidamente, usando solamente cuatro pasos por regla general cuando tiene éxito y tres cuando se estanca (no está mal para un espacio de estados con $8^8 \approx 17$ millones de estados).

El algoritmo de la Figura 4.11 se para si alcanza una meseta donde el mejor sucesor tiene el mismo valor que el estado actual. ¿Podría ser una buena idea no continuar (y permitir un **movimiento lateral** con la esperanza de que la meseta realmente sea una terraza, como se muestra en la Figura 4.10)? La respuesta es por lo general sí, pero debemos tener cuidado. Si siempre permitimos movimientos laterales cuando no hay ningún movimiento ascendente, va a ocurrir un bucle infinito siempre que el algoritmo alcance un máximo local plano que no sea una terraza. Una solución común es poner un límite sobre el número de movimientos consecutivos laterales permitidos. Por ejemplo, podríamos permitir hasta, digamos, 100 movimientos laterales consecutivos en el problema de las ocho reinas. Esto eleva el porcentaje de casos de problemas resueltos por la ascensión de

MOVIMIENTO LATERAL

Figura 4.13 Ilustración de por qué las crestas causan dificultades para la ascensión de colinas. La rejilla de estados (círculos oscuros) se pone sobre una cresta que se eleva de izquierda a derecha y crea una secuencia de máximos locales que no están directamente relacionados el uno con el otro. De cada máximo local, todas las acciones disponibles se señalan cuesta abajo.

colinas del 14 al 94 por ciento. El éxito viene con un coste: el algoritmo hace en promedio aproximadamente 21 pasos para cada caso satisfactorio y 64 para cada fracaso.

ASCENSIÓN DE COLINAS ESTOCÁSTICA

ASCENSIÓN DE COLINAS DE PRIMERA OPCIÓN

Se han inventado muchas variantes de la ascensión de colinas. La **ascensión de colinas estocástica** escoge aleatoriamente de entre los movimientos ascendentes; la probabilidad de selección puede variar con la pendiente del movimiento ascendente. Éste por lo general converge más despacio que la subida más escarpada, pero en algunos paisajes de estados encuentra mejores soluciones. La **ascensión de colinas de primera opción** implementa una ascensión de colinas estocástica generando sucesores al azar hasta que se genera uno que es mejor que el estado actual. Esta es una buena estrategia cuando un estado tiene muchos (por ejemplo, miles) sucesores. El ejercicio 4.16 le pide investigar esto.

ASCENSIÓN DE COLINAS DE REINICIO ALEATORIO

Los algoritmos de ascensión de colinas descritos hasta ahora son incompletos, a menudo dejan de encontrar un objetivo, cuando éste existe, debido a que pueden estancarse sobre máximos locales. **La ascensión de colinas de reinicio aleatorio** adopta el refrán conocido, «si al principio usted no tiene éxito, intente, intente otra vez». Esto conduce a una serie de búsquedas en ascensión de colinas desde los estados iniciales generados aleatoriamente[8], parándose cuando se encuentra un objetivo. Es completa con probabilidad acercándose a 1, por la razón trivial de que generará finalmente un estado objetivo como el estado inicial. Si cada búsqueda por ascensión de colinas tiene una probabilidad p de éxito, entonces el número esperado de reinicios requerido es $1/p$. Para ejemplos de ocho reinas sin permitir movimientos laterales, $p \approx 0{,}14$, entonces necesitamos aproximadamente siete iteraciones para encontrar un objetivo (seis fracasos y un éxito). El número esperado de pasos es el coste de una iteración acertada más $(1 - p)/p$ veces el coste de fracaso, o aproximadamente 22 pasos. Cuando permitimos movimientos laterales, son necesarios $1/0{,}94 \approx 1{,}06$ iteraciones por regla general y $(1 \times 21) + (0{,}06/0{,}94) \times 64 \approx 25$ pasos. Para las ocho reinas, entonces, la ascensión de colinas de reinicio aleatorio es muy eficaz. Incluso para tres millones de reinas, la aproximación puede encontrar soluciones en menos de un minuto[9].

El éxito de la ascensión de colinas depende muchísimo de la forma del paisaje del espacio de estados: si hay pocos máximos locales y mesetas, la ascensión de colinas con reinicio aleatorio encontrará una solución buena muy rápidamente. Por otro lado, muchos problemas reales tienen un paisaje que parece más bien una familia de puerco espines sobre un suelo llano, con puerco espines en miniatura que viven en la punta de cada aguja del puerco espín, y así *indefinidamente*. Los problemas NP-duros típicamente tienen un número exponencial de máximos locales. A pesar de esto, un máximo local, razonablemente bueno, a menudo se puede encontrar después de un número pequeño de reinicios.

Búsqueda de temple simulado

Un algoritmo de ascensión de colinas que nunca hace movimientos «cuesta abajo» hacia estados con un valor inferior (o coste más alto) garantiza ser incompleto, porque puede

[8] La generación de un estado *aleatorio* de un espacio de estados especificado implícitamente puede ser un problema difícil en sí mismo.

[9] Luby *et al*. (1993) demuestran que es mejor, a veces, reiniciar un algoritmo de búsqueda aleatoria después de cierta cantidad fija de tiempo y que puede ser mucho más eficiente que permitir que continúe la búsqueda indefinidamente. Rechazar o limitar el número de movimientos laterales es un ejemplo de esto.

estancarse en un máximo local. En contraste, un camino puramente aleatorio, es decir moviéndose a un sucesor elegido uniformemente aleatorio de un conjunto de sucesores, es completo, pero sumamente ineficaz. Por lo tanto, parece razonable intentar combinar la ascensión de colinas con un camino aleatorio de algún modo que produzca tanto eficacia como completitud. El **temple simulado** es ese algoritmo. En metalurgia, el **temple** es el proceso utilizado para templar o endurecer metales y cristales calentándolos a una temperatura alta y luego gradualmente enfriarlos, así permite al material fundirse en un estado cristalino de energía baja. Para entender el temple simulado, cambiemos nuestro punto de vista de la ascensión de colinas al **gradiente descendente** (es decir, minimizando el coste) e imaginemos la tarea de colocar una pelota de ping-pong en la grieta más profunda en una superficie desigual. Si dejamos solamente rodar a la pelota, se parará en un mínimo local. Si sacudimos la superficie, podemos echar la pelota del mínimo local. El truco es sacudir con bastante fuerza para echar la pelota de mínimos locales, pero no lo bastante fuerte para desalojarlo del mínimo global. La solución del temple simulado debe comenzar sacudiendo con fuerza (es decir, a una temperatura alta) y luego gradualmente reducir la intensidad de la sacudida (es decir, a más baja temperatura).

TEMPLE SIMULADO

GRADIENTE DESCENDENTE

El bucle interno del algoritmo del temple simulado (Figura 4.14) es bastante similar a la ascensión de colinas. En vez de escoger el *mejor* movimiento, sin embargo, escoge un movimiento *aleatorio*. Si el movimiento mejora la situación, es siempre aceptado. Por otra parte, el algoritmo acepta el movimiento con una probabilidad menor que uno. La probabilidad se disminuye exponencialmente con la «maldad» de movimiento (la cantidad ΔE por la que se empeora la evaluación). La probabilidad también disminuye cuando «la temperatura» T baja: los «malos» movimientos son más probables al comienzo cuando la temperatura es alta, y se hacen más improbables cuando T disminuye. Uno puede demostrar que si el *esquema* disminuye T bastante despacio, el algoritmo encontrará un óptimo global con probabilidad cerca de uno.

```
función TEMPLE-SIMULADO(problema, esquema) devuelve un estado solución
  entradas: problema, un problema
            esquema, una aplicación desde el tiempo a «temperatura»
  variables locales: actual, un nodo
                     siguiente, un nodo
                     T, una «temperatura» controla la probabilidad de un paso hacia abajo

  actual ← HACER-NODO(ESTADO-INICIAL[problema])
  para t ← 1 a ∞ hacer
    T ← esquema[t]
    si T = 0 entonces devolver actual
    siguiente ← un sucesor seleccionado aleatoriamente de actual
    ΔE ← VALOR[siguiente] − VALOR[actual]
    si ΔE > 0 entonces actual ← siguiente
    en caso contrario actual ← siguiente sólo con probabilidad e^(ΔE/T)
```

Figura 4.14 Algoritmo de búsqueda de temple simulado, una versión de la ascensión de colinas estocástico donde se permite descender a algunos movimientos. Los movimientos de descenso se aceptan fácilmente al comienzo en el programa de templadura y luego menos, conforme pasa el tiempo. La entrada del *esquema* determina el valor de T como una función de tiempo.

A principios de los años 80, el temple simulado fue utilizado ampliamente para resolver problemas de distribución VLSI. Se ha aplicado ampliamente a programación de una fábrica y otras tareas de optimización a gran escala. En el Ejercicio 4.16, le pedimos que compare su funcionamiento con el de la ascensión de colinas con reinicio aleatorio sobre el puzle de las *n*-reinas.

Búsqueda por haz local

BÚSQUEDA POR HAZ LOCAL

Guardar solamente un nodo en la memoria podría parecer una reacción extrema para el problema de limitaciones de memoria. El algoritmo[10] de **búsqueda por haz local** guarda la pista de k estados (no sólo uno). Comienza con estados generados aleatoriamente. En cada paso, se generan todos los sucesores de los k estados. Si alguno es un objetivo, paramos el algoritmo. Por otra parte, se seleccionan los k mejores sucesores de la lista completa y repetimos.

A primera vista, una búsqueda por haz local con k estados podría parecerse a ejecutar k reinicios aleatorios en paralelo en vez de en secuencia. De hecho, los dos algoritmos son bastantes diferentes. En una búsqueda de reinicio aleatorio, cada proceso de búsqueda se ejecuta independientemente de los demás. *En una búsqueda por haz local, la información útil es pasada entre los k hilos paralelos de búsqueda.* Por ejemplo, si un estado genera varios sucesores buenos y los otros $k - 1$ estados generan sucesores malos, entonces el efecto es que el primer estado dice a los demás, «¡Venid aquí, la hierba es más verde!» El algoritmo rápidamente abandona las búsquedas infructuosas y mueve sus recursos a donde se hace la mayor parte del progreso.

BÚSQUEDA DE HAZ ESTOCÁSTICA

En su forma más simple, la búsqueda de haz local puede sufrir una carencia de diversidad entre los k estados (se pueden concentrar rápidamente en una pequeña región del espacio de estados, haciendo de la búsqueda un poco más que una versión cara de la ascensión de colinas). Una variante llamada **búsqueda de haz estocástica**, análoga a la ascensión de colinas estocástica, ayuda a aliviar este problema. En vez de elegir los k mejores del conjunto de sucesores candidatos, la búsqueda de haz estocástica escoge a k sucesores aleatoriamente, con la probabilidad de elegir a un sucesor como una función creciente de su valor. La búsqueda de haz estocástica muestra algún parecido con el proceso de selección natural, por lo cual los «sucesores» (descendientes) de un «estado» (organismo) pueblan la siguiente generación según su «valor» (idoneidad o salud).

Algoritmos genéticos

ALGORITMO GENÉTICO

Un **algoritmo genético** (o AG) es una variante de la búsqueda de haz estocástica en la que los estados sucesores se generan combinando *dos* estados padres, más que modificar un solo estado. La analogía a la selección natural es la misma que con la búsqueda de haz estocástica, excepto que ahora tratamos con reproducción sexual más que con la reproducción asexual.

[10] Búsqueda por haz local es una adaptación de la **búsqueda de haz**, que es un algoritmo basado en camino.

POBLACIÓN

INDIVIDUO

Como en la búsqueda de haz, los AGs comienzan con un conjunto de k estados generados aletaoriamente, llamados **población**. Cada estado, o **individuo**, está representado como una cadena sobre un alfabeto finito (el más común, una cadenas de 0s y 1s). Por ejemplo, un estado de las ocho reinas debe especificar las posiciones de las ocho reinas, cada una en una columna de ocho cuadrados, y se requieren $8 \times \log_2 8 = 24$ bits. O bien, el estado podría representarse como ocho dígitos, cada uno en el rango de uno a ocho (veremos más tarde que las dos codificaciones se comportan de forman diferente). La Figura 4.15(a) muestra una población de cuatro cadenas de ocho dígitos que representan estados de ocho reinas.

FUNCIÓN IDONEIDAD

En la Figura 4.15(b)-(e) se muestra la producción de la siguiente generación de estados. En (b) cada estado se tasa con la función de evaluación o (en terminología AG) la **función idoneidad**. Una función de idoneidad debería devolver valores más altos para estados mejores, así que, para el problema de las 8-reinas utilizaremos el número de pares de reinas no atacadas, que tiene un valor de 28 para una solución. Los valores de los cuatro estados son 24, 23, 20 y 11. En esta variante particular del algoritmo genético, la probabilidad de ser elegido para la reproducción es directamente proporcional al resultado de idoneidad, y los porcentajes se muestran junto a los tanteos.

CRUCE

En (c), se seleccionan dos pares, de manera aleatoria, para la reproducción, de acuerdo con las probabilidades en (b). Notemos que un individuo se selecciona dos veces y uno ninguna[11]. Para que cada par se aparee, se elige aleatoriamente un punto de **cruce** de las posiciones en la cadena. En la Figura 4.15 los puntos de cruce están después del tercer dígito en el primer par y después del quinto dígito en el segundo par[12].

En (d), los descendientes se crean cruzando las cadenas paternales en el punto de cruce. Por ejemplo, el primer hijo del primer par consigue los tres primeros dígitos del

Figura 4.15 Algoritmo genético. La población inicial en (a) es ordenada con la función idoneidad en (b), y resultan pares para acoplamiento en (c). Ellos producen los descendientes en (d), que están sujetos a mutación en (e).

[11] Hay muchas variantes de esta regla de selección. Puede demostrarse que el método **selectivo**, en el que se desechan todos los individuos debajo de un umbral dado, converge más rápido que la versión aleatoria (Basum *et al.*, 1995).

[12] Son aquí los asuntos de codificación. Si se usa una codificación de 24 bit en vez de ocho dígitos, entonces el punto de cruce tiene 2/3 de posibilidad de estar en medio de un dígito, que resulta en una mutación esencialmente arbitraria de ese dígito.

primer padre y los dígitos restantes del segundo padre, mientras que el segundo hijo consigue los tres primeros dígitos del segundo padre y el resto del primer padre. En la Figura 4.16 se muestran los estados de las ocho reinas implicados en este paso de reproducción. El ejemplo ilustra el hecho de que, cuando dos estados padres son bastante diferentes, la operación de cruce puede producir un estado que está lejos de cualquiera de los estados padre. Esto es, a menudo, lo que ocurre al principio del proceso en el que la población es bastante diversa, así que el cruce (como en el temple simulado) con frecuencia realiza pasos grandes, al principio, en el espacio de estados en el proceso de búsqueda y pasos más pequeños, más tarde, cuando la mayor parte de individuos son bastante similares.

Figura 4.16 Los estados de las ocho reinas correspondientes a los dos primeros padres de la Figura 4.15(c) y el primer descendiente de Figura 4.15(d). Las columnas sombreadas se pierden en el paso de la transición y las columnas no sombreadas se mantienen.

MUTACIÓN

Finalmente, en (e), cada posición está sujeta a la **mutación** aleatoria con una pequeña probabilidad independiente. Un dígito fue transformado en el primer, tercer, y cuarto descendiente. El problema de las 8-reinas corresponde a escoger una reina aleatoriamente y moverla a un cuadrado aleatorio en su columna. La figura 4.17 describe un algoritmo que implementa todos estos pasos.

Como en la búsqueda por haz estocástica, los algoritmos genéticos combinan una tendencia ascendente con exploración aleatoria y cambian la información entre los hilos paralelos de búsqueda. La ventaja primera, si hay alguna, del algoritmo genético viene de la operación de cruce. Aún puede demostrarse matemáticamente que, si las posiciones del código genético se permutan al principio en un orden aleatorio, el cruce no comunica ninguna ventaja. Intuitivamente, la ventaja viene de la capacidad del cruce para combinar bloques grandes de letras que han evolucionado independientemente para así realizar funciones útiles, de modo que se aumente el nivel de granularidad en el que funciona la búsqueda. Por ejemplo, podría ser que poner las tres primeras reinas en posiciones 2, 4 y 6 (donde ellas no se atacan las unas a las otras) constituya un bloque útil que pueda combinarse con otros bloques para construir una solución.

ESQUEMA

La teoría de los algoritmos genéticos explica cómo esta teoría trabaja utilizando la idea de un **esquema**, una subcadena en la cual algunas de las posiciones se pueden dejar inespecíficas. Por ejemplo, el esquema 246***** describe todos los estados de

```
función ALGORITMO-GENÉTICO(población,IDONEIDAD) devuelve un individuo
  entradas: población, un conjunto de individuos
            IDONEIDAD, una función que mide la capacidad de un individuo

  repetir
    nueva_población ← conjunto vacío
    bucle para i desde 1 hasta TAMAÑO(población) hacer
      x ← SELECCIÓN-ALEATORIA(población, IDONEIDAD)
      y ← SELECCIÓN-ALEATORIA(población, IDONEIDAD)
      hijo ← REPRODUCIR(x,y)
      si (probabilidad aleatoria pequeña) entonces hijo ← MUTAR(hijo)
      añadir hijo a nueva_problación
    población ← nueva_población
  hasta que algún individuo es bastante adecuado, o ha pasado bastante tiempo
  devolver el mejor individuo en la población, de acuerdo con la IDONEIDAD
```

```
función REPRODUCIR(x,y) devuelve un individuo
  entradas: x,y, padres individuales

  n ← LONGITUD(x)
  c ← número aleatorio de 1 a n
  devolver AÑADIR(SUBCADENA(x, 1, c),SUBCADENA(y, c + 1, n))
```

Figura 4.17 Algoritmo genético. El algoritmo es el mismo que el de la Figura 4.15, con una variación: es la versión más popular; cada cruce de dos padres produce sólo un descendiente, no dos.

ocho reinas en los cuales las tres primeras reinas están en posiciones 2, 4 y 6 respectivamente. A las cadenas que emparejan con el esquema (tal como 24613578) se les llaman **instancias** del esquema. Se puede demostrar que, si la idoneidad media de las instancias de un esquema está por encima de la media, entonces el número de instancias del esquema dentro de la población crecerá con el tiempo. Claramente, este efecto improbablemente será significativo si los bits adyacentes están totalmente no relacionados uno al otro, porque entonces habrá pocos bloques contiguos que proporcionen un beneficio consistente. Los algoritmos genéticos trabajan mejor cuando los esquemas corresponden a componentes significativos de una solución. Por ejemplo, si la cadenas son una representación de una antena, entonces los esquemas pueden representar los componentes de la antena, tal como reflectores y deflectores. Un componente bueno probablemente estará bien en una variedad de diseños diferentes. Esto sugiere que el uso acertado de algoritmos genéticos requiere la ingeniería cuidadosa de la representación.

En la práctica, los algoritmos genéticos han tenido un impacto extendido sobre problemas de optimización, como disposición de circuitos y el programado del trabajo en tiendas. Actualmente, no está claro si lo solicitado de los algoritmos genéticos proviene de su funcionamiento o de sus orígenes estéticamente agradables de la teoría de la evolución. Se han hecho muchos trabajos para identificar las condiciones bajo las cuales los algoritmos genéticos funcionan bien.

EVOLUCIÓN Y BÚSQUEDA

La teoría de la **evolución** fue desarrollada por Charles Darwin (1859) en *El Origen de Especies por medio de la Selección Natural*. La idea central es simple: las variaciones (conocidas como **mutaciones**) ocurren en la reproducción y serán conservadas en generaciones sucesivas aproximadamente en la proporción de su efecto sobre la idoneidad reproductiva.

La teoría de Darwin fue desarrollada sin el conocimiento de cómo los rasgos de los organismos se pueden heredar y modificar. Las leyes probabilísticas que gobiernan estos procesos fueron identificadas primero por Gregor Mendel (1866), un monje que experimentó con guisantes dulces usando lo que él llamó la fertilización artificial. Mucho más tarde, Watson y Crick (1953) identificaron la estructura de la molécula de ADN y su alfabeto, AGTC (adenina, guanina, timina, citocina). En el modelo estándar, la variación ocurre tanto por mutaciones en la secuencia de letras como por «el cruce» (en el que el ADN de un descendiente se genera combinando secciones largas del ADN de cada padre).

Ya se ha descrito la analogía con algoritmos de búsqueda local; la diferencia principal entre la búsqueda de haz estocástica y la evolución es el uso de la reproducción sexual, en donde los sucesores se generan a partir de *múltiples* organismos más que de solamente uno. Los mecanismos actuales de la evolución son, sin embargo, mucho más ricos de lo que permiten la mayoría de los algoritmos genéticos. Por ejemplo, las mutaciones pueden implicar inversiones, copias y movimientos de trozos grandes de ADN; algunos virus toman prestado el ADN de un organismo y lo insertan en otro; y hay genes reemplazables que no hacen nada pero se copian miles de veces dentro del genoma. Hay hasta genes que envenenan células de compañeros potenciales que no llevan el gen, bajando el aumento de sus posibilidades de réplica. Lo más importante es el hecho de que los *genes codifican los mecanismos* por los cuales se reproduce y traslada el genoma en un organismo. En algoritmos genéticos, esos mecanismos son un programa separado que no está representado dentro de las cadenas manipuladas.

La evolución Darwiniana podría parecer más bien un mecanismo ineficaz, y ha generado ciegamente aproximadamente 10^{45} organismos sin mejorar su búsqueda heurística un ápice. 50 años antes de Darwin, sin embargo, el gran naturalista francés Jean Lamarck (1809) propuso una teoría de evolución por la cual los rasgos *adquiridos por la adaptación durante la vida de un organismo* serían pasados a su descendiente. Tal proceso sería eficaz, pero no parece ocurrir en la naturaleza. Mucho más tarde, James Baldwin (1896) propuso una teoría superficialmente similar: aquel comportamiento aprendido durante la vida de un organismo podría acelerar la evolución. A diferencia de la de Lamarck, la teoría de Baldwin es completamente consecuente con la evolución Darwiniana, porque confía en presiones de selección que funcionan sobre individuos que han encontrado óptimos locales entre el conjunto de comportamientos posibles permitidos por su estructura genética. Las simulaciones por computadores modernos confirman que «el efecto de Baldwin» es real, a condición de que la evolución «ordinaria» pueda crear organismos cuya medida de rendimiento está, de alguna manera, correlacionada con la idoneidad actual.

4.4 Búsqueda local en espacios continuos

En el Capítulo 2, explicamos la diferencia entre entornos discretos y continuos, señalando que la mayor parte de los entornos del mundo real son continuos. Aún ninguno de los algoritmos descritos puede manejar espacios de estados continuos, ¡la función sucesor en la mayor parte de casos devuelve infinitamente muchos estados! Esta sección proporciona una *muy breve* introducción a técnicas de búsqueda local para encontrar soluciones óptimas en espacios continuos. La literatura sobre este tema es enorme; muchas de las técnicas básicas se originaron en el siglo XVII, después del desarrollo de cálculo Newton y Leibniz[13]. Encontraremos usos para estas técnicas en varios lugares del libro, incluso en los capítulos sobre aprendizaje, visión y robótica. En resumen, cualquier cosa que trata con el mundo real.

Comencemos con un ejemplo. Supongamos que queremos colocar tres nuevos aeropuertos en cualquier lugar de Rumanía, de forma tal que la suma de las distancias al cuadrado de cada ciudad sobre el mapa (Figura 3.2) a su aeropuerto más cercano sea mínima. Entonces el espacio de estados está definido por las coordenadas de los aeropuertos: (x_1, y_1), (x_2, y_2), y (x_3, y_3). Es un espacio *seis-dimensional*; también decimos que los estados están definidos por seis **variables** (en general, los estados están definidos por un vector n-dimensional de variables, $\mathbf{x}$). Moverse sobre este espacio se corresponde a movimientos de uno o varios de los aeropuertos sobre el mapa. La función objetivo $f(x_1, y_1, x_2, y_2, x_3, y_3)$ es relativamente fácil calcularla para cualquier estado particular una vez que tenemos las ciudades más cercanas, pero bastante complicado anotar en general.

Un modo de evitar problemas continuos es simplemente discretizar la vecindad de cada estado. Por ejemplo, podemos movernos sólo sobre un aeropuerto a la vez, en la dirección x o y, en una cantidad fija $\pm\delta$. Con seis variables, nos da 12 sucesores para cada estado. Podemos aplicar entonces cualquiera de los algoritmos de búsqueda local descritos anteriormente. Uno puede aplicar también la ascensión de colinas estocástica y el temple simulado directamente, sin discretizar el espacio. Estos algoritmos eligen a los sucesores aleatoriamente, que pueden hacerse por la generación de vectores aleatorios de longitud δ.

GRADIENTE

Hay muchos métodos que intentan usar el **gradiente** del paisaje para encontrar un máximo. El gradiente de la función objetivo es un vector ∇f que nos da la magnitud y la dirección de la inclinación más escarpada. Para nuestro problema, tenemos

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

En algunos casos, podemos encontrar un máximo resolviendo la ecuación $\nabla f = 0$ (esto podría hacerse, por ejemplo, si estamos colocando solamente un aeropuerto; la solución es la media aritmética de todas las coordenadas de las ciudades). En muchos casos, sin embargo, esta ecuación no puede resolverse de forma directa. Por ejemplo, con tres aeropuertos, la expresión para el gradiente depende de qué ciudades son las más cercanas a cada aeropuerto en el estado actual. Esto significa que podemos calcular el gradiente

[13] Un conocimiento básico de cálculo multivariante y aritmética vectorial es útil cuando uno lee esta sección.

localmente pero no *globalmente*. Incluso, podemos realizar todavía la ascensión de colinas por la subida más escarpada poniendo al día el estado actual con la fórmula

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

donde α es una constante pequeña. En otros casos, la función objetivo podría no estar disponible de una forma diferenciable, por ejemplo, el valor de un conjunto particular de posiciones de los aeropuertos puede determinarse ejecutando algún paquete de simulación económica a gran escala. En esos casos, el llamado **gradiente empírico** puede determinarse evaluando la respuesta a pequeños incrementos y decrecimientos en cada coordenada. La búsqueda de gradiente empírico es la misma que la ascensión de colinas con subida más escarpada en una versión discretizada del espacio de estados.

GRADIENTE EMPÍRICO

Bajo la frase «α es una constante pequeña» se encuentra una enorme variedad de métodos ajustando α. El problema básico es que, si α es demasiado pequeña, necesitamos demasiados pasos; si α es demasiado grande, la búsqueda podría pasarse del máximo. La técnica de **línea de búsqueda** trata de vencer este dilema ampliando la dirección del gradiente actual (por lo general duplicando repetidamente α) hasta que f comience a disminuir otra vez. El punto en el cual esto ocurre se convierte en el nuevo estado actual. Hay varias escuelas de pensamiento sobre cómo debe elegirse la nueva dirección en este punto.

LÍNEA DE BÚSQUEDA

Para muchos problemas, el algoritmo más eficaz es el venerable método de **Newton-Raphson** (Newton, 1671; Raphson, 1690). Es una técnica general para encontrar raíces de funciones, es decir la solución de ecuaciones de la forma $g(x) = 0$. Trabaja calculando una nueva estimación para la raíz x según la fórmula de Newton.

NEWTON-RAPHSON

$$x \leftarrow x - g(x)/g'(x)$$

Para encontrar un máximo o mínimo de f, tenemos que encontrar x tal que el gradiente es cero (es decir, $\nabla f(\mathbf{x}) = \mathbf{0}$). Así $g(x)$, en la fórmula de Newton, se transforma en $\nabla f(\mathbf{x})$, y la ecuación de actualización puede escribirse en forma de vector-matriz como

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x})\, \nabla f(\mathbf{x})$$

donde $\mathbf{H}_f(\mathbf{x})$ es la matriz **Hesiana** de segundas derivadas, cuyo los elementos H_{ij} están descritos por $\partial^2 f/\partial x_i \partial x_j$. Ya que el Hesiano tiene n^2 entradas, Newton-Raphson se hace costoso en espacios dimensionalmente altos, y por tanto, se han desarrollado muchas aproximaciones.

HESIANA

Los métodos locales de búsqueda sufren de máximos locales, crestas, y mesetas tanto en espacios de estados continuos como en espacios discretos. Se pueden utilizar el reinicio aleatorio y el temple simulado y son a menudo provechosos. Los espacios continuos dimensionalmente altos son, sin embargo, lugares grandes en los que es fácil perderse.

Un tema final, que veremos de pasada, es la **optimización con restricciones**. Un problema de optimización está restringido si las soluciones debieran satisfacer algunas restricciones sobre los valores de cada variable. Por ejemplo, en nuestro problema de situar aeropuertos, podría restringir los lugares para estar dentro de Rumanía y sobre la tierra firme (más que en medio de lagos). La dificultad de los problemas de optimización con restricciones depende de la naturaleza de las restricciones y la función objetivo. La categoría más conocida es la de los problemas de **programación lineal**, en los cuales las restricciones deben ser desigualdades lineales formando una región *convexa* y la función

OPTIMIZACIÓN CON RESTRICCIONES

PROGRAMACIÓN LINEAL

objetiva es también lineal. Los problemas de programación lineal pueden resolverse en tiempo polinomial en el número de variables. También se han estudiado problemas con tipos diferentes de restricciones y funciones objetivo (programación cuadrática, programación cónica de segundo orden, etcétera).

4.5 Agentes de búsqueda *online* y ambientes desconocidos

BÚSQUEDA *OFFLINE*

Hasta ahora nos hemos centrado en agentes que usan algoritmos de **búsqueda *offline***. Ellos calculan una solución completa antes de poner un pie en el mundo real (*véase* la Figura 3.1), y luego ejecutan la solución sin recurrir a su percepciones. En contraste, un agente de **búsqueda en línea (*online*)**[14] funciona **intercalando** el cálculo y la acción: primero toma una acción, entonces observa el entorno y calcula la siguiente acción. La búsqueda *online* es una buena idea en dominios dinámicos o semidinámicos (dominios donde hay una penalización por holgazanear y por utilizar demasiado tiempo para calcular). La búsqueda *online* es una idea incluso mejor para dominios estocásticos. En general, una búsqueda *offline* debería presentar un plan de contingencia exponencialmente grande que considere todos los acontecimientos posibles, mientras que una búsqueda *online* necesita sólo considerar lo que realmente pasa. Por ejemplo, a un agente que juega al ajedrez se le aconseja que haga su primer movimiento mucho antes de que se haya resuelto el curso completo del juego.

BÚSQUEDA *ONLINE*

PROBLEMA DE EXPLORACIÓN

La búsqueda online es una idea *necesaria* para un **problema de exploración**, donde los estados y las acciones son desconocidos por el agente; un agente en este estado de ignorancia debe usar sus acciones como experimentos para determinar qué hacer después, y a partir de ahí debe intercalar el cálculo y la acción.

El ejemplo básico de búsqueda *online* es un robot que se coloca en un edificio nuevo y lo debe explorar para construir un mapa, que puede utilizar para ir desde A a B. Los métodos para salir de laberintos (conocimiento requerido para aspirar a ser héroe de la Antigüedad) son también ejemplos de algoritmos de búsqueda *online*. Sin embargo, la exploración espacial no es la única forma de la exploración. Considere a un bebé recién nacido: tiene muchas acciones posibles, pero no sabe los resultados de ellas, y ha experimentado sólo algunos de los estados posibles que puede alcanzar. El descubrimiento gradual del bebé de cómo trabaja el mundo es, en parte, un proceso de búsqueda *online*.

Problemas de búsqueda en línea (*online*)

Un problema de búsqueda *online* puede resolverse solamente por un agente que ejecute acciones, más que por un proceso puramente computacional. Asumiremos que el agente sabe lo siguiente:

[14] El término «en línea (*online*)» es comúnmente utilizado en informática para referirse a algoritmos que deben tratar con los datos de entrada cuando se reciben, más que esperar a que esté disponible el conjunto entero de datos de entrada.

- ACCIONES (s), que devuelve una lista de acciones permitidas en el estado s;
- Funciones de coste individual $c(s, a, s')$ (notar que no puede usarse hasta que el agente sepa que s' es el resultado); y
- TEST-OBJETIVO(s).

Notemos en particular que el agente *no puede* tener acceso a los sucesores de un estado excepto si intenta realmente todas las acciones en ese estado. Por ejemplo, en el problema del laberinto de la Figura 4.18, el agente no sabe que *Subir* desde (1,1) conduce a (1,2); ni, habiendo hecho esto, sabe que *Bajar* lo devolverá a (1,1). Este grado de ignorancia puede reducirse en algunas aplicaciones (para ejemplo, un robot explorador podría saber cómo trabajan sus acciones de movimiento y ser ignorante sólo de las posiciones de los obstáculos).

Asumiremos que el agente puede reconocer siempre un estado que ha visitado anteriormente, y asumiremos que las acciones son deterministas (en el Capítulo 17, se relajarán estos dos últimos axiomas). Finalmente, el agente podría tener acceso a una función heurística admisible $h(s)$ que estime la distancia del estado actual a un estado objetivo. Por ejemplo, en la Figura 4.18, el agente podría saber la posición del objetivo y ser capaz de usar la distancia heurística de Manhattan.

Típicamente, el objetivo del agente es alcanzar un estado objetivo minimizando el coste (otro objetivo posible es explorar simplemente el entorno entero). El costo es el costo total del camino por el que el agente viaja realmente. Es común comparar este costo con el costo del camino que el agente seguiría *si supiera el espacio de búsqueda de antemano,* es decir, el camino más corto actual (o la exploración completa más corta). En el lenguaje de algoritmos *online*, se llama **proporción competitiva**; nos gustaría que fuera tan pequeña como sea posible.

PROPORCIÓN COMPETITIVA

Aunque ésta suene como una petición razonable, es fácil ver que la mejor proporción alcanzable competitiva es infinita en algunos casos. Por ejemplo, si algunas acciones son irreversibles, la búsqueda *online* podría alcanzar, por casualidad, un estado sin salida del cual no es accesible ningún estado objetivo.

Quizás encuentre el término «por casualidad» poco convincente (después de todo, podría haber un algoritmo que no tome el camino sin salida mientras explora). Nuestra

Figura 4.18 Un problema sencillo de un laberinto. El agente comienza en S y debe alcanzar G, pero no sabe nada del entorno.

reclamación, para ser más precisos, consiste en que *ningún algoritmo puede evitar callejones sin salida en todos los espacios de estados*. Considere los dos espacios de estados sin salida de la Figura 4.19 (a). A un algoritmo de búsqueda *online* que haya visitado los estados *S* y *A*, los dos espacios de estados parecen *idénticos*, entonces debe tomar la misma decisión en ambos. Por lo tanto, fallará en uno de ellos. Es un ejemplo de un **argumento de adversario** (podemos imaginar un adversario que construye el espacio de estados, mientras el agente lo explora, y puede poner el objetivo y callejones sin salida donde le guste).

ARGUMENTO DE ADVERSARIO

Los callejones sin salida son una verdadera dificultad para la exploración de un robot (escaleras, rampas, acantilados, y todas las clases de posibilidades presentes en terrenos naturales de acciones irreversibles). Para avanzar, asumiremos simplemente que el espacio de estados es **seguramente explorable**, es decir, algún estado objetivo es alcanzable desde cualquier estado alcanzable. Los espacios de estados con acciones reversibles, como laberintos y 8-puzles, pueden verse como grafos no-dirigidos y son claramente explorables.

SEGURAMENTE EXPLORABLE

Incluso en entornos seguramente explorables no se puede garantizar ninguna proporción competitiva acotada si hay caminos de costo ilimitado. Esto es fácil de demostrar en entornos con acciones irreversibles, pero de hecho permanece cierto también para el caso reversible, como se muestra en la Figura 4.19(b). Por esta razón, es común describir el funcionamiento de los algoritmos de búsqueda *online* en términos del tamaño del espacio de estados entero más que, solamente, por la profundidad del objetivo más superficial.

Figura 4.19 (a) Dos espacios de estados que podrían conducir a un agente de búsqueda *online* a un callejón sin salida. Cualquier agente fallará en al menos uno de estos espacios. (b) Un entorno de dos-dimensiones que puede hacer que un agente de búsqueda *online* siga una ruta arbitrariamente ineficaz al objetivo. Ante cualquier opción que tome el agente, el adversario bloquea esa ruta con otra pared larga y delgada, de modo que el camino seguido sea mucho más largo que el camino mejor posible.

Agentes de búsqueda en línea (*online*)

Después de cada acción, un agente *online* recibe una percepción al decirle que estado ha alcanzado; de esta información, puede aumentar su mapa del entorno. El mapa actual se usa para decidir dónde ir después. Esta intercalación de planificación y acción significa que los algoritmos de búsqueda *online* son bastante diferentes de los algoritmos de búsqueda *offline* vistos anteriormente. Por ejemplo, los algoritmos *offline* como A* tienen la capacidad de expandir un nodo en una parte del espacio y luego inmediatamente expandir un nodo en otra parte del espacio, porque la expansión de un nodo implica simulación más que verdaderas acciones. Un algoritmo *online*, por otra parte, puede expandir sólo el nodo que ocupa físicamente. Para evitar viajar a través de todo el árbol para expandir el siguiente nodo, parece mejor expandir los nodos en un orden *local*. La búsqueda primero en profundidad tiene exactamente esta propiedad, porque (menos cuando volvemos hacia atrás) el siguiente nodo a expandir es un hijo del nodo anteriormente expandido.

En la Figura 4.20 se muestra un agente de búsqueda primero en profundidad *online*. Este agente almacena su mapa en una tabla, *resultado*[*a*,*s*], que registra el estado que resulta de ejecutar la acción *a* en el estado *s*. Siempre que una acción del estado actual no haya sido explorada, el agente intenta esa acción. La dificultad viene cuando el agente ha intentado todas las acciones en un estado. En la búsqueda primero en profundidad *offline*, el estado es simplemente quitado de la cola; en una búsqueda *online*, el agente tiene que volver atrás físicamente. La búsqueda primero en profundidad, significa volver al estado el cual el agente incorporó el estado actual más recientemente. Esto se con-

```
función AGENTE-BPP-ONLINE(s′) devuelve una acción
  entradas: s′, una percepción que identifica el estado actual
  estático: resultado, una tabla, indexada por la acción y el estado, inicialmente vacía
            noexplorados, una tabla que enumera, para cada estado visitado, las acciones
              todavía no intentadas
            nohaciatras, una tabla que enumera, para cada estado visitado, los nodos hacia
              atrás todavía no intentados
            s,a, el estado y acción previa, inicialmente nula

  si TEST-OBJETIVO(s′) entonces devolver parar
  si s′ es un nuevo estado entonces noexplorados[s′] ← ACCIONES(s′)
  si s es no nulo entonces hacer
    resultado[a,s] ← s′
    añadir s al frente de nohaciatras[s′]
  si noexplorados[s′] esta vacío entonces
    si nohaciatras[s′] esta vacío entonces devolver parar
    en caso contrario a ← una acción b tal que resultado[b, s′] = POP(nohaciatras[s′])
  en caso contrario a ← POP(noexplorados[s′])
  s ← s′
  devolver a
```

Figura 4.20 Un agente de búsqueda *online* que utiliza la exploración primero en profundidad. El agente es aplicable, solamente, en espacios de búsqueda bidireccionales.

sigue guardando una tabla que pone en una lista, para cada estado, los estados predecesores a los cuales aún no ha vuelto. Si el agente se ha quedado sin estados a los que volver, entonces su búsqueda se ha completado.

Recomendamos al lector que compruebe el progreso del AGENTE-BPP-ONLINE cuando se aplica al laberinto de la Figura 4.18. Es bastante fácil ver que el agente, en el caso peor, terminará por cruzar cada enlace en el espacio de estados exactamente dos veces. Por exploración, esto es lo óptimo; para encontrar un objetivo, por otra parte, la proporción competitiva del agente podría ser arbitrariamente mala si se viaja sobre un camino largo cuando hay un objetivo directamente al lado del estado inicial. Una variante *online* de la profundidad iterativa resuelve este problema; para un entorno representado por un árbol uniforme, la proporción competitiva del agente es una constante pequeña.

A causa de su método de vuelta atrás, el AGENTE-BPP-ONLINE trabaja sólo en espacios de estados donde las acciones son reversibles. Hay algoritmos ligeramente más complejos que trabajan en espacios de estados generales, pero ninguno de estos algoritmos tiene una proporción competitiva acotada.

Búsqueda local en línea (*online*)

Como la búsqueda primero en profundidad, la **búsqueda de ascensión de colinas** tiene la propiedad de localidad en sus expansiones de los nodos. ¡De hecho, porque mantiene un estado actual en memoria, la búsqueda de ascensión de colinas es ya un algoritmo de búsqueda *online*! Desafortunadamente, no es muy útil en su forma más simple porque deja al agente que se sitúe en máximos locales con ningún movimiento que hacer. Por otra parte, los reinicios aleatorios no pueden utilizarse, porque el agente no puede moverse a un nuevo estado.

CAMINO ALEATORIO

En vez de reinicios aleatorios, podemos considerar el uso de un **camino aleatorio** para explorar el entorno. Un camino aleatorio selecciona simplemente al azar una de las acciones disponibles del estado actual; se puede dar preferencia a las acciones que todavía no se han intentado. Es fácil probar que un camino aleatorio encontrará *al final* un objetivo o termina su exploración, a condición de que el espacio sea finito[15]. Por otra parte, el proceso puede ser muy lento. La Figura 4.21 muestra un entorno en el que un

Figura 4.21 Un entorno en el cual un camino aleatorio utilizará exponencialmente muchos pasos para encontrar el objetivo.

[15] El caso infinito es mucho más difícil. ¡Los caminos aleatorios son completos en rejillas unidimensionales y de dos dimensiones infinitas, pero no en rejillas tridimensionales! En el último caso, la probabilidad de que el camino vuelva siempre al punto de partida es alrededor de 0,3405 (*véase* a Hughes, 1995, para una introducción general).

camino aleatorio utilizará un número exponencial de pasos para encontrar el objetivo, porque, en cada paso, el progreso hacia atrás es dos veces más probable que el progreso hacia delante. El ejemplo es artificial, por supuesto, pero hay muchos espacios de estados del mundo real cuya topología causa estas clases de «trampas» para los caminos aleatorios.

Aumentar a la ascensión de colinas con *memoria* más que aleatoriedad, resulta ser una aproximación más eficaz. La idea básica es almacenar una «mejor estimación actual» $H(s)$ del coste para alcanzar el objetivo desde cada estado que se ha visitado. El comienzo de $H(s)$ es justo la estimación heurística $h(s)$ y se actualiza mientras que el agente gana experiencia en el espacio de estados. La Figura 4.22 muestra un ejemplo sencillo en un espacio de estados unidimensional. En (a), el agente parece estar estancado en un mínimo local plano en el estado sombreado. Más que permanecer donde está, el agente debe seguir por donde parece ser la mejor trayectoria al objetivo, basada en las estimaciones de los costes actuales para sus vecinos. El coste estimado para alcanzar el objetivo a través de un vecino s' es el coste para conseguir s' más el coste estimado para conseguir un objetivo desde ahí, es decir, $c(s, a, s') + H(s')$. En el ejemplo, hay dos acciones con costos estimados $1 + 9$ y $1 + 2$, así parece que lo mejor posible es moverse a la derecha. Ahora, está claro que la estimación del costo de dos para el estado sombreado fue demasiado optimista. Puesto que el mejor movimiento costó uno y condujo a un estado que está al menos a dos pasos de un objetivo, el estado sombreado debe estar por lo menos a tres pasos de un objetivo, así que su H debe actualizarse adecuadamente, como se muestra en la figura 4.22(b). Continuando este proceso, el agente se moverá hacia delante y hacia atrás dos veces más, actualizando H cada vez y «apartando» el mínimo local hasta que se escapa a la derecha.

Figura 4.22 Cinco iteraciones de AA*TR en un espacio de estados unidimensional. Cada estado se marca con $H(s)$, la estimación del costo actual para alcanzar un objetivo, y cada arco se marca con su costo. El estado sombreado marca la ubicación del agente, y se rodean los valores actualizados en cada iteración.

AA*TR

En la Figura 4.23 se muestra un agente que implementa este esquema, llamado aprendiendo A* en tiempo real (**AA*TR**). Como el AGENTE-BPP-ONLINE, éste construye un mapa del entorno usando la tabla *resultado*. Actualiza el costo estimado para el estado que acaba de dejar y entonces escoge el movimiento «aparentemente mejor» según sus costos estimados actuales. Un detalle importante es que las acciones que todavía no se han intentado en un estado s siempre se supone que dirigen inmediatamente al objetivo con el costo menor posible, $h(s)$. Este **optimismo bajo la incertidumbre** anima al agente a explorar nuevos y posiblemente caminos prometedores.

OPTIMISMO BAJO INCERTIDUMBRE

```
función AGENTE-AA*TR(s′) devuelve una acción
  entradas: s′, una percepción que identifica el estado actual
  estático: resultado, una tabla, indexada por la acción y el estado, inicialmente vacía
            H, una tabla de costos estimados indexada por estado, inicialmente vacía
            s, a, el estado y acción previa, inicialmente nula

  si TEST-OBJETIVO(s′) entonces devolver parar
  si s′ es un nuevo estado (no en H) entonces H[s′] ← h(s′)
  a menos que s sea nulo
    resultado[a, s] ← s′
    H[s] ← min_{b ∈ ACCIONES(s)} COSTO-AA*TR(s, b, resultado[b, s],H)
  a ← una acción b de ACCIONES(s′) que minimiza COSTO-AA*TR(s′, b, resultado[b, s′],H)
  s ← s′
  devolver a

función COSTO-AA*TR(s, a, s′, H) devuelve un costo estimado
  si s′ está indefinido entonces devolver h(s)
  en otro caso devolver c(s, a, s′) + H[s′]
```

Figura 4.23 El AGENTE-AA*TR escoge una acción según los valores de los estados vecinos, que se actualizan conforme el agente se mueve sobre el espacio de estados.

Un agente AA*TR garantiza encontrar un objetivo en un entorno seguramente explorable y finito. A diferencia de A*, sin embargo, no es completo para espacios de estados infinitos (hay casos donde se puede dirigir infinitamente por mal camino). Puede explorar un entorno de n estados en $O(n^2)$ pasos, en el caso peor, pero a menudo lo hace mejor. El agente AA*TR es sólo uno de una gran familia de agentes *online* que pueden definirse especificando la regla de la selección de la acción y que actualiza la regla de maneras diferentes. Discutiremos esta familia, que fue desarrollada originalmente para entornos estocásticos, en el Capítulo 21.

Aprendizaje en la búsqueda en línea (*online*)

La ignorancia inicial de los agentes de búsqueda *online* proporciona varias oportunidades para aprender. Primero, los agentes aprenden un «mapa» del entorno (más precisamente, el resultado de cada acción en cada estado) simplemente registrando cada una de sus experiencias (notemos que la suposición de entornos deterministas quiere decir que

una experiencia es suficiente para cada acción). Segundo, los agentes de búsqueda locales adquieren estimaciones más exactas del valor de cada estado utilizando las reglas de actualización local, como en AA*TR. En el Capítulo 21 veremos que éstas actualizaciones convergen finalmente a valores *exactos* para cada estado, con tal de que el agente explore el espacio de estados de manera correcta. Una vez que se conocen los valores exactos, se pueden tomar las decisiones óptimas simplemente moviéndose al sucesor con el valor más alto (es decir, la ascensión de colinas pura es entonces una estrategia óptima).

Si usted siguió nuestra sugerencia de comprobar el comportamiento del AGENTE-BPP-ONLINE en el entorno de la Figura 4.18, habrá advertido que el agente no es muy brillante. Por ejemplo, después de ver que la acción *Arriba* va de (1,1) a (1,2), el agente no tiene la menor idea todavía que la acción *Abajo* vuelve a (1,1), o que la acción *Arriba* va también de (2,1) a (2,2), de (2,2) a (2,3), etcétera. En general, nos gustaría que el agente aprendiera que *Arriba* aumenta la coordenada *y* a menos que haya una pared en el camino, que hacia *Abajo* la reduce, etcétera. Para que esto suceda, necesitamos dos cosas: primero, necesitamos una representación formal y explícitamente manipulable para estas clases de reglas generales; hasta ahora, hemos escondido la información dentro de la caja negra llamada función sucesor. La Parte III está dedicada a este tema. Segundo, necesitamos algoritmos que puedan construir reglas generales adecuadas a partir de la observación específica hecha por el agente. Estos algoritmos se tratan en el Capítulo 18.

4.6 Resumen

Este capítulo ha examinado la aplicación de **heurísticas** para reducir los costos de la búsqueda. Hemos mirado varios algoritmos que utilizan heurísticas y encontramos que la optimalidad tiene un precio excesivo en términos del costo de búsqueda, aún con heurísticas buenas.

- **Búsqueda primero el mejor** es una BÚSQUEDA-GRAFO donde los nodos no expandidos de costo mínimo (según alguna medida) se escogen para la expansión. Los algoritmos primero el mejor utilizan típicamente una función heurística $h(n)$ que estima el costo de una solución desde n.
- **Búsqueda primero el mejor avara** expande nodos con $h(n)$ mínima. No es óptima, pero es a menudo eficiente.
- **Búsqueda A*** expande nodos con mínimo $f(n) = g(n) + h(n)$. A* es completa y óptima, con tal que garanticemos que $h(n)$ sea admisible (para BÚSQUEDA-ARBOL) o consistente (para BÚSQUEDA-GRAFO). La complejidad en espacio de A* es todavía prohibitiva.
- El rendimiento de los algoritmos de búsqueda heurística depende de la calidad de la función heurística. Las heurísticas buenas pueden construirse a veces relajando la definición del problema, por costos de solución precalculados para sub-problemas en un modelo de bases de datos, o aprendiendo de la experiencia con clases de problemas.

- **BRPM** y **A*MS** son algoritmos de búsqueda robustos y óptimos que utilizan cantidades limitadas de memoria; con suficiente tiempo, pueden resolver los problemas que A* no puede resolver porque se queda sin memoria.
- Los métodos de *búsqueda local,* como la **ascensión de colinas,** operan en formulaciones completas de estados, manteniendo sólo un número pequeño de nodos en memoria. Se han desarrollado varios algoritmos estocásticos, inclusive el **temple simulado**, que devuelven soluciones óptimas cuando se da un apropiado programa de enfriamiento. Muchos métodos de búsqueda local se pueden utilizar también para resolver problemas en espacios continuos.
- Un **algoritmo genético** es una búsqueda de ascensión de colinas estocástica en la que se mantiene una población grande de estados. Los estados nuevos se generan por **mutación** y por **cruce**, combinando pares de estados de la población.
- Los **problemas de exploración** surgen cuando el agente no tiene la menor idea acerca de los estados y acciones de su entorno. Para entornos seguramente explorables, los agentes de **búsqueda en línea** pueden construir un mapa y encontrar un objetivo si existe. Las estimaciones de las heurística, que se actualizan por la experiencia, proporcionan un método efectivo para escapar de mínimos locales.

Notas bibliográficas e históricas

El uso de información heurística en la resolución de problemas aparece en un artículo de Simon y Newell (1958), pero la frase «búsqueda heurística» y el uso de las funciones heurísticas que estiman la distancia al objetivo llegaron algo más tarde (Newell y Ernst, 1965; Lin, 1965). Doran y Michie (1966) dirigieron muchos estudios experimentales de búsqueda heurística aplicados a varios problemas, especialmente al 8-puzle y 15-puzle. Aunque Doran y Michie llevaran a cabo un análisis teórico de la longitud del camino y «penetrancia» (proporción entre la longitud del camino y el número total de nodos examinados hasta el momento) en la búsqueda heurística, parecen haber ignorado la información proporcionada por la longitud actual del camino. El algoritmo A*, incorporando la longitud actual del camino en la búsqueda heurística, fue desarrollado por Hart, Nilsson y Raphael (1968), con algunas correcciones posteriores (Hart *et al.*, 1972). Dechter y Pearl (1985) demostraron la eficiencia óptima de A*.

El artículo original de A* introdujo la condición de consistencia en funciones heurísticas. La condición de monotonía fue introducida por Pohl (1977) como un sustituto más sencillo, pero Pearl (1984) demostró que las dos eran equivalentes. Varios algoritmos precedentes de A* utilizaron el equivalente de listas abiertas y cerradas; éstos incluyen la búsqueda primero en anchura, primero en profundidad, y costo uniforme (Bellman, 1957; Dijkstra, 1959). El trabajo de Bellman en particular mostró la importancia de añadir los costos de los caminos para simplificar los algoritmos de optimización.

Pohl (1970, 1977) fue el pionero en el estudio de la relación entre el error en las funciones heurísticas y la complejidad en tiempo de A*. La demostración de que A* se ejecuta en un tiempo lineal si el error de la función heurística está acotado por una constante puede encontrarse en Pohl (1977) y en Gaschnig (1979). Pearl (1984) reforzó este re-

sultado para permitir un crecimiento logarítmico en el error. El «factor de ramificación eficaz», medida de la eficiencia de la búsqueda heurística, fue propuesto por Nilsson (1971).

Hay muchas variaciones del algoritmo A*. Pohl (1973) propuso el uso del *ponderado dinámico*, el cual utiliza una suma ponderada $f_w(n) = w_g g(n) + w_h h(n)$ de la longitud del camino actual y de la función heurística como una función de evaluación, más que la suma sencilla $f(n) = g(n) + h(n)$ que utilizó A*. Los pesos w_g y w_h se ajustan dinámicamente con el progreso de la búsqueda. Se puede demostrar que el algoritmo de Pohl es ϵ-admisible (es decir, garantiza encontrar las soluciones dentro de un factor $1 + \epsilon$ de la solución óptima) donde ϵ es un parámetro suministrado al algoritmo. La misma propiedad es exhibida por el algoritmo A^*_ϵ (Pearl, 1984), el cual puede escoger cualquier nodo de la franja tal que su *f*-costo esté dentro de un factor $1 + \epsilon$ del nodo de la franja de *f*-costo más pequeño. La selección se puede hacer para minimizar el costo de la búsqueda.

A* y otros algoritmos de búsqueda en espacio de estados están estrechamente relacionados con las técnicas de *ramificar-y-acotar* ampliamente utilizadas en investigación operativa (Lawler y Wood, 1966). Las relaciones entre la búsqueda en espacio de estados y ramificar-y-acotar se han investigado en profundidad (Kumar y Kanal, 1983; Nau *et al*., 1984; Kumas *et al*., 1988). Martelli y Montanari (1978) demostraron una conexión entre la programación dinámica (*véase* el Capítulo 17) y cierto tipo de búsqueda en espacio de estados. Kumar y Kanal (1988) intentan una «ambiciosa unificación» de la búsqueda heurística, programación dinámica, y técnicas de ramifica-y-acotar bajo el nombre de PDC (el «proceso de decisión compuesto»).

Como los computadores a finales de los años 1950 y principios de los años 1960 tenían como máximo unas miles de palabras de memoria principal, la búsqueda heurística con memoria-acotada fue un tema de investigación. El Grafo Atravesado (Doran y Michie, 1966), uno de los programas de búsqueda más antiguos, compromete a un operador después de realizar una búsqueda primero el mejor hasta el límite de memoria. A*PI (Korf, 1985a, 1985b) fue el primero que usó un algoritmo de búsqueda heurística, óptima, con memoria-acotada y de la que se han desarrollado un número grande de variantes. Un análisis de la eficiencia de A*PI y de sus dificultades con las heurística real-valoradas aparece en Patrick *et al*. (1992).

El BRPM (Korf, 1991, 1993) es realmente algo más complicado que el algoritmo mostrado en la Figura 4.5, el cual está más cercano a un algoritmo, desarrollado independientemente, llamado **extensión iterativa**, o EI (Russell, 1992). BRPM usa una cota inferior y una cota superior; los dos algoritmos se comportan idénticamente con heurísticas admisibles, pero BRPM expande nodos en orden primero el mejor hasta con una heurística inadmisible. La idea de guardar la pista del mejor camino alternativo apareció en la implementación elegante, en Prolog, de A* realizada por Bratko (1986) y en el algoritmo DTA* (Russell y Wefald, 1991). El trabajo posterior también habló de espacios de estado meta nivel y aprendizaje meta nivel.

EXTENSIÓN ITERATIVA

El algoritmo A*M apareció en Chakrabarti *et al*. (1989). A*MS, o A*M simplificado, surgió de una tentativa de implementación de A*M como un algoritmo de comparación para IE (Russell, 1992). Kaindl y Khorsand (1994) han aplicado A*MS para producir un algoritmo de búsqueda bidireccional considerablemente más rápido que los

algoritmos anteriores. Korf y Zhang (2000) describen una aproximación divide-y-vencerás, y Zhou y Hansen (2002) introducen una búsqueda A* en un grafo de memoria-acotada. Korf (1995) revisa las técnicas de búsqueda de memoria-acotada.

La idea de que las heurísticas admisibles pueden obtenerse por relajación del problema aparece en el trabajo seminal de Held y Karp (1970), quien utilizó la heurística del mínimo-atravesando el árbol para resolver el PVC (*véase* el Ejercicio 4.8).

La automatización del proceso de relajación fue implementado con éxito por Prieditis (1993), construido sobre el trabajo previo de Mostow (Mostow y Prieditis, 1989). El uso del modelo de bases de datos para obtener heurísticas admisibles se debe a Gasser (1995) y Culberson y Schaeffer (1998); el modelo de bases de datos disjuntas está descrito por Korf y Felner (2002). La interpretación probabilística de las heurística fue investigada en profundidad por Pearl (1984) y Hansson y Mayer (1989).

La fuente bibliográfica más comprensiva sobre heurísticas y algoritmos de búsqueda heurísticos está en el texto *Heuristics* de Pearl (1984). Este libro cubre de una manera especialmente buena la gran variedad de ramificaciones y variaciones de A*, incluyendo demostraciones rigurosas de sus propiedades formales. Kanal y Kumar (1988) presentan una antología de artículos importantes sobre la búsqueda heurística. Los nuevos resultados sobre algoritmos de búsqueda aparecen con regularidad en la revista *Artificial Intelligence*.

Las técnicas locales de búsqueda tienen una larga historia en matemáticas y en informática. En efecto, el método de Newton-Raphson (Newton, 1671; Raphson, 1690) puede verse como un método de búsqueda local muy eficiente para espacios continuos en los cuales está disponible la información del gradiente. Brent (1973) es una referencia clásica para algoritmos de optimización que no requieren tal información. La búsqueda de haz, que hemos presentado como un algoritmo de búsqueda local, se originó como una variante de anchura-acotada de la programación dinámica para el reconocimiento de la voz en el sistema HARPY (Lowerre, 1976). En Pearl (1984, el Capítulo 5) se analiza en profundidad un algoritmo relacionado.

El tema de la búsqueda local se ha fortalecido en los últimos años por los resultados sorprendentemente buenos en problemas de satisfacción de grandes restricciones como las *n*-reinas (Minton *et al*., 1992) y de razonamiento lógico (Selman *et al*., 1992) y por la incorporación de aleatoriedad, múltiples búsquedas simultáneas, y otras mejoras. Este renacimiento, de lo que Christos Papadimitriou ha llamado algoritmos de la «Nueva Era», ha provocado también el interés entre los informáticos teóricos (Koutsoupias y Papadimitriou, 1992; Aldous y Vazirani, 1994). En el campo de la investigación operativa, una variante de la ascensión de colinas, llamada **búsqueda tabú,** ha ganado popularidad (Glover, 1989; Glover y Laguna, 1997). Realizado sobre modelos de memoria limitada a corto plazo de los humanos, este algoritmo mantienen una lista tabú de *k* estados, previamente visitados, que no pueden visitarse de nuevo; así se mejora la eficiencia cuando se busca en grafos y además puede permitir que el algoritmo se escape de algunos mínimos locales. Otra mejora útil sobre la ascensión de colinas es el algoritmo de STAGE (Boyan y Moore, 1998). La idea es usar los máximos locales encontrados por la ascensión de colinas de reinicio aleatorio para conseguir una idea de la forma total del paisaje. El algoritmo adapta una superficie suave al conjunto de máximos locales y luego calcula analíticamente el máximo global de esa superficie. Éste se convierte en el nuevo punto de

BÚSQUEDA TABÚ

reinicio. Se ha demostrado que este algoritmo trabaja, en la práctica, sobre problemas difíciles. (Gomes *et al*., 1998) mostraron que las distribuciones, en tiempo de ejecución, de los algoritmos de vuelta atrás sistemáticos a menudo tienen una **distribución de cola pesada**, la cual significa que la probabilidad de un tiempo de ejecución muy largo es mayor que lo que sería predicho si los tiempos de ejecución fueran normalmente distribuidos. Esto proporciona una justificación teórica para los reinicios aleatorios.

DISTRIBUCIÓN DE COLA PESADA

El temple simulado fue inicialmente descrito por Kirkpatrick *et al*. (1983), el cual se basó directamente en el **algoritmo de Metrópolis** (usado para simular sistemas complejos en la física (Metrópolis *et al*., 1953) y fue supuestamente inventado en la cena Los Alamos). El temple simulado es ahora un campo en sí mismo, con cien trabajos publicados cada año.

Encontrar soluciones óptimas en espacios continuos es la materia de varios campos, incluyendo la **teoría de optimización**, **teoría de control óptima**, y el **cálculo de variaciones**. Los convenientes (y prácticos) puntos de entrada son proporcionados por Press *et al*. (2002) y Bishop (1995). La **programación lineal** (PL) fue una de las primeras aplicaciones de computadores; el **algoritmo simplex** (Wood y Dantzig, 1949; Dantzig, 1949) todavía se utiliza a pesar de la complejidad exponencial, en el peor caso. Karmarkar (1984) desarrolló un algoritmo de tiempo polinomial práctico para PL.

El trabajo de Sewal Wright (1931), sobre el concepto de la **idoneidad de un paisaje**, fue un precursor importante para el desarrollo de los algoritmos genéticos. En los años 50, varios estadísticos, incluyendo Box (1957) y Friedman (1959), usaron técnicas evolutivas para problemas de optimización, pero no fue hasta que Rechenberg (1965, 1973) introdujera las **estrategias de evolución** para resolver problemas de optimización para planos aerodinámicos en la que esta aproximación ganó popularidad. En los años 60 y 70, John Holland (1975) defendió los algoritmos genéticos, como un instrumento útil y como un método para ampliar nuestra comprensión de la adaptación, biológica o de otra forma (Holland, 1995). El movimiento de **vida artificial** (Langton, 1995) lleva esta idea un poco más lejos, viendo los productos de los algoritmos genéticos como *organismos* más que como soluciones de problemas. El trabajo de Hinton y Nowlan (1987) y Ackley y Littman (1991) en este campo ha hecho mucho para clarificar las implicaciones del efecto de Baldwin. Para un tratamiento más a fondo y general sobre la evolución, recomendamos a Smith y Szathmáry (1999).

ESTRATEGIAS DE EVOLUCIÓN

VIDA ARTIFICIAL

La mayor parte de comparaciones de los algoritmos genéticos con otras aproximaciones (especialmente ascensión de colinas estocástica) han encontrado que los algoritmos genéticos son más lentos en converger (O'Reilly y Oppacher, 1994; Mitchell *et al*., 1996; Juels y Watternberg, 1996; Baluja, 1997). Tales conclusiones no son universalmente populares dentro de la comunidad de AG, pero tentativas recientes dentro de esa comunidad, para entender la búsqueda basada en la población como una forma aproximada de aprendizaje Bayesiano (*véase* el Capítulo 20), quizás ayude a cerrar el hueco entre el campo y sus críticas (Pelikan *et al*., 1999). La teoría de **sistemas dinámicos cuadráticos** puede explicar también el funcionamiento de AGs (Rabani *et al*., 1998). *Véase* Lohn *et al*. (2001) para un ejemplo de AGs aplicado al diseño de antenas, y Larrañaga *et al*. (1999) para una aplicación al problema de viajante de comercio.

PROGRAMACIÓN GENÉTICA

El campo de la **programación genética** está estrechamente relacionado con los algoritmos genéticos. La diferencia principal es que las representaciones, que son muta-

das y combinadas, son programas más que cadenas de bits. Los programas se representan en forma de árboles de expresión; las expresiones pueden estar en un lenguaje estándar como Lisp o pueden estar especialmente diseñadas para representar circuitos, controladores del robot, etcétera. Los cruces implican unir los subárboles más que las subcadenas. De esta forma, la mutación garantiza que los descendientes son expresiones gramaticalmente correctas, que no lo serían si los programas fueran manipulados como cadenas.

El interés reciente en la programación genética fue estimulado por el trabajo de John Koza (Koza, 1992, 1994), pero va por detrás de los experimentos con código máquina de Friedberg (1958) y con autómatas de estado finito de Fogel *et al.* (1966). Como con los algoritmos genéticos, hay un debate sobre la eficacia de la técnica. Koza *et al.* (1999) describen una variedad de experimentos sobre el diseño automatizado de circuitos de dispositivos utilizando la programación genética.

Las revistas *Evolutionary Computation* y *IEEE Transactions on Evolutionary Computation* cubren los algoritmos genéticos y la programación genética; también se encuentran artículos en *Complex Systems*, *Adaptative Behavior*, y *Artificial Life*. Las conferencias principales son la *International Conference on Genetic Algorithms* y la *Conference on Genetic Programming*, recientemente unidas para formar la *Genetic and Evolutionary Computation Conference*. Los textos de Melanie Mitchell (1996) y David Fogel (2000) dan descripciones buenas del campo.

Los algoritmos para explorar espacios de estados desconocidos han sido de interés durante muchos siglos. La búsqueda primero en profundidad en un laberinto puede implementarse manteniendo la mano izquierda sobre la pared; los bucles pueden evitarse marcando cada unión. La búsqueda primero en profundidad falla con acciones irreversibles; el problema más general de exploración de **grafos Eulerianos** (es decir, grafos en los cuales cada nodo tiene un número igual de arcos entrantes y salientes) fue resuelto por un algoritmo debido a Hierholzer (1873). El primer estudio cuidadoso algorítmico del problema de exploración para grafos arbitrarios fue realizado por Deng y Papadimitriou (1990), quienes desarrollaron un algoritmo completamente general, pero demostraron que no era posible una proporción competitiva acotada para explorar un grafo general. Papadimitriou y Yannakakis (1991) examinaron la cuestión de encontrar caminos a un objetivo en entornos geométricos de planificación de caminos (donde todas las acciones son reversibles). Ellos demostraron que es alcanzable una pequeña proporción competitiva con obstáculos cuadrados, pero no se puede conseguir una proporción acotada con obstáculos generales rectangulares (*véase* la Figura 4.19).

GRAFOS EULERIANOS

El algoritmo LRTA* fue desarrollado por Korf (1990) como parte de una investigación en la **búsqueda en tiempo real** para entornos en los cuales el agente debe actuar después de buscar en sólo una cantidad fija del tiempo (una situación mucho más común en juegos de dos jugadores). El LRTA* es, de hecho, un caso especial de algoritmos de aprendizaje por refuerzo para entornos estocásticos (Barto *et al.*, 1995). Su política de optimismo bajo incertidumbre (siempre se dirige al estado no visitado más cercano) puede causar un modelo de exploración que es menos eficiente, en el caso sin información, que la búsqueda primero en profundidad simple (Koening, 2000). Dasgupta *et al.* (1994) mostraron que la búsqueda en profundidad iterativa *online* es óptimamente eficiente para encontrar un objetivo en un árbol uniforme sin la información heurís-

BÚSQUEDA EN TIEMPO REAL

tica. Algunas variantes informadas sobre el tema LRTA* se han desarrollado con métodos diferentes para buscar y actualizar dentro de la parte conocida del grafo (Pemberton y Korf, 1992). Todavía no hay una buena comprensión de cómo encontrar los objetivos con eficiencia óptima cuando se usa información heurística.

BÚSQUEDA PARALELA

El tema de los algoritmos de **búsqueda paralela** no se ha tratado en el capítulo, en parte porque requiere una discusión larga de arquitecturas paralelas de computadores. La búsqueda paralela llega a ser un tema importante en IA y en informática teórica. Una introducción breve a la literatura de IA se puede encontrar en Mahanti y Daniels (1993).

EJERCICIOS

4.1 Trace cómo opera la búsqueda A* aplicada al problema de alcanzar Bucarest desde Lugoj utilizando la heurística distancia en línea recta. Es decir, muestre la secuencia de nodos que considerará el algoritmo y los valores f, g, y h para cada nodo.

4.2 El **algoritmo de camino heurístico** es una búsqueda primero el mejor en la cual la función objetivo es $f(n) = (2 - w)g(n) + wh(n)$. ¿Para qué valores del w está garantizado que el algoritmo sea óptimo? ¿Qué tipo de búsqueda realiza cuando $w = 0$?, ¿cuándo $w = 1$? y ¿cuándo $w = 2$?

4.3 Demuestre cada una de las declaraciones siguientes:

a) La búsqueda primero en anchura es un caso especial de la búsqueda de coste uniforme.

b) La búsqueda primero en anchura, búsqueda primero en profundidad, y la búsqueda de coste uniforme son casos especiales de la búsqueda primero el mejor.

c) La búsqueda de coste uniforme es un caso especial de la búsqueda A*.

4.4 Idee un espacio de estados en el cual A*, utilizando la BÚSQUEDA-GRAFO, devuelva una solución sub-óptima con una función $h(n)$ admisible pero inconsistente.

4.5 Vimos en la página 109 que la heurística de distancia en línea recta dirige la búsqueda primero el mejor voraz por mal camino en el problema de ir de Iasi a Fagaras. Sin embargo, la heurística es perfecta en el problema opuesto: ir de Fagaras a Iasi. ¿Hay problemas para los cuales la heurística engaña en ambas direcciones?

4.6 Invente una función heurística para el 8-puzle que a veces sobrestime, y muestre cómo puede conducir a una solución subóptima sobre un problema particular (puede utilizar un computador para ayudarse). Demuestre que, si h nunca sobrestima en más de c, A*, usando h, devuelve una solución cuyo coste excede de la solución óptima en no más de c.

4.7 Demuestre que si una heurística es consistente, debe ser admisible. Construya una heurística admisible que no sea consistente.

4.8 El problema del viajante de comercio (PVC) puede resolverse con la heurística del árbol mínimo (AM), utilizado para estimar el coste de completar un viaje, dado que ya se ha construido un viaje parcial. El coste de AM del conjunto de ciudades es

la suma más pequeña de los costos de los arcos de cualquier árbol que une todas las ciudades.

a) Muestre cómo puede obtenerse esta heurística a partir de una versión relajada del PVC.
b) Muestre que la heurística AM domina la distancia en línea recta.
c) Escriba un generador de problemas para ejemplos del PVC donde las ciudades están representadas por puntos aleatorios en el cuadrado unidad.
d) Encuentre un algoritmo eficiente en la literatura para construir el AM, y úselo con un algoritmo de búsqueda admisible para resolver los ejemplos del PVC.

4.9 En la página 122, definimos la relajación del 8-puzle en el cual una ficha podía moverse del cuadrado A al cuadrado B si B era el blanco. La solución exacta de este problema define la **heurística de Gaschnig** (Gaschnig, 1979). Explique por qué la heurística de Gaschnig es al menos tan exacta como h_1 (fichas mal colocadas), y muestre casos donde es más exacta que h_1 y que h_2 (distancia de Manhattan). ¿Puede sugerir un modo de calcular la heurística de Gaschnig de manera eficiente?

4.10 Dimos dos heurísticas sencillas para el 8-puzle: distancia de Manhattan y fichas mal colocadas. Varias heurísticas en la literatura pretendieron mejorarlas, *véase*, por ejemplo, Nilsson (1971), Mostow y Prieditis (1989), y Hansson *et al.* (1992). Pruebe estas mejoras, implementando las heurísticas y comparando el funcionamiento de los algoritmos que resultan.

4.11 Dé el nombre del algoritmo que resulta de cada uno de los casos siguientes:

a) Búsqueda de haz local con $k = 1$.
b) Búsqueda de haz local con $k = \infty$.
c) Temple simulado con $T = 0$ en cualquier momento.
d) Algoritmo genético con tamaño de la población $N = 1$.

4.12 A veces no hay una función de evaluación buena para un problema, pero hay un método de comparación bueno: un modo de decir si un nodo es mejor que el otro, sin adjudicar valores numéricos. Muestre que esto es suficiente para hacer una búsqueda primero el mejor. ¿Hay un análogo de A*?

4.13 Relacione la complejidad en tiempo de LRTA* con su complejidad en espacio.

4.14 Suponga que un agente está en un laberinto de 3x3 como el de la Figura 4.18. El agente sabe que su posición inicial es (1,1), que el objetivo está en (3,3), y que las cuatro acciones *Arriba, Abajo, Izquierda, Derecha* tienen sus efectos habituales a menos que estén bloqueadas por una pared. El agente *no* sabe dónde están las paredes internas. En cualquier estado, el agente percibe el conjunto de acciones legales; puede saber también si el estado ha sido visitado antes o si es un nuevo estado.

a) Explique cómo este problema de búsqueda *online* puede verse como una búsqueda *offline* en el espacio de estados de creencia, donde el estado de creencia inicial incluye todas las posibles configuraciones del entorno. ¿Cómo de grande es el estado de creencia inicial? ¿Cómo de grande es el espacio de estados de creencia?
b) ¿Cuántas percepciones distintas son posibles en el estado inicial?

c) Describa las primeras ramas de un plan de contingencia para este problema. ¿Cómo de grande (aproximadamente) es el plan completo?

Nótese que este plan de contingencia es una solución para *todos los entornos posibles* que encajan con la descripción dada. Por lo tanto, intercalar la búsqueda y la ejecución no es estrictamente necesario hasta en entornos desconocidos.

4.15 En este ejercicio, exploraremos el uso de los métodos de búsqueda local para resolver los PVCs del tipo definido en el Ejercicio 4.8.

a) Idee una aproximación de la ascensión de colinas para resolver los PVSs. Compare los resultados con soluciones óptimas obtenidas con el algoritmo A* con la heurística AM (Ejercicio 4.8).

b) Idee una aproximación del algoritmo genético al problema del viajante de comercio. Compare los resultados a las otras aproximaciones. Puede consultar Larrañaga *et al*. (1999) para algunas sugerencias sobre las representaciones.

4.16 Genere un número grande de ejemplos del 8-puzle y de las 8-reinas y resuélvalos (donde sea posible) por la ascensión de colinas (variantes de subida más escarpada y de la primera opción), ascensión de colinas con reinicio aleatorio, y temple simulado. Mida el coste de búsqueda y el porcentaje de problemas resueltos y represente éstos gráficamente contra el costo óptimo de solución. Comente sus resultados.

4.17 En este ejercicio, examinaremos la ascensión de colinas en el contexto de navegación de un robot, usando el entorno de la Figura 3.22 como un ejemplo.

a) Repita el Ejercicio 3.16 utilizando la ascensión de colinas. ¿Cae alguna vez su agente en un mínimo local? ¿Es *posible* con obstáculos convexos?

b) Construya un entorno no convexo poligonal en el cual el agente cae en mínimos locales.

c) Modifique el algoritmo de ascensión de colinas de modo que, en vez de hacer una búsqueda a profundidad 1 para decidir dónde ir, haga una búsqueda a profundidad-k. Debería encontrar el mejor camino de k-pasos y hacer un paso sobre el camino, y luego repetir el proceso.

d) ¿Hay algún k para el cual esté garantizado que el nuevo algoritmo se escape de mínimos locales?

e) Explique cómo LRTA* permite al agente escaparse de mínimos locales en este caso.

4.18 Compare el funcionamiento de A* y BRPM sobre un conjunto de problemas generados aleatoriamente en dominios del 8-puzle (con distancia de Manhattan) y del PVC (con AM, *véase* el Ejercicio 4.8). Discuta sus resultados. ¿Qué le pasa al funcionamiento de la BRPM cuando se le añade un pequeño número aleatorio a los valores heurísticos en el dominio del 8-puzle?

5 Problemas de satisfacción de restricciones

En donde veremos cómo el tratar los estados como más que sólo pequeñas cajas negras conduce a la invención de una variedad de nuevos poderosos métodos de búsqueda y a un entendimiento más profundo de la estructura y complejidad del problema.

Los Capítulos 3 y 4 exploraron la idea de que los problemas pueden resolverse buscando en un espacio de **estados**. Estos estados pueden evaluarse con heurísticas específicas del dominio y probados para ver si son estados objetivo. Desde el punto de vista del algoritmo de búsqueda, sin embargo, cada estado es una **caja negra** sin la estructura perceptible interna. Se representa por una estructura de datos arbitraria a la que se puede acceder sólo con las rutinas *específicas de problema* (la función sucesor, función heurística, y el test objetivo).

CAJA NEGRA

Este capítulo examina **problemas de satisfacción de restricciones**, cuyos estados y test objetivo forman una **representación** muy simple, estándar y estructurada (Sección 5.1). Los algoritmos de búsqueda se pueden definir aprovechándose de la estructura de los estados y utilizan las heurísticas de *propósito general* más que heurísticas *específicas de problema* para así permitir la solución de problemas grandes (Secciones 5.2-5.3). Quizá lo más importante sea que la representación estándar del test objetivo revela la estructura del problema (Sección 5.4). Esto conduce a métodos de descomposición de problemas y a una comprensión de la conexión entre la estructura de un problema y la dificultad para resolverlo.

PROBLEMA DE SATISFACCIÓN DE RESTRICCIONES

REPRESENTACIÓN

5.1 Problemas de satisfacción de restricciones

VARIABLES

RESTRICCIONES

Formalmente, un **problema de satisfacción de restricciones** (o PSR) está definido por un conjunto de **variables**, $X_1, X_2 \ldots, X_n$, y un conjunto de **restricciones**, $C_1, C_2 \ldots, C_m$. Cada

DOMINIO

VALORES

ASIGNACIÓN

CONSISTENTE

FUNCIÓN OBJETIVO

variable X_i tiene un **dominio** no vacío D_i de **valores** posibles. Cada restricción C_i implica algún subconjunto de variables y especifica las combinaciones aceptables de valores para ese subconjunto. Un estado del problema está definido por una **asignación** de valores a unas o todas las variables, $\{X_i = v_i, X_j = v_j \ldots\}$. A una asignación que no viola ninguna restricción se llama asignación **consistente** o legal. Una asignación completa es una asignación en la que se menciona cada variable, y una **solución** de un PSR es una asignación completa que satisface todas las restricciones. Algunos problemas de satisfacción de restricciones (PSRs) también requieren una solución que maximiza una **función objetivo**.

¿Qué significa todo esto? Suponga que, cansados de Rumanía, miramos un mapa de Australia que muestra cada uno de sus estados y territorios, como en la Figura 5.1(a), y que nos encargan la tarea de colorear cada región de rojo, verde o azul de modo que ninguna de las regiones vecinas tenga el mismo color. Para formularlo como un PSR, definimos las variables de las regiones: *AO, TN, Q, NGS, V, AS* y *T*. El dominio de cada variable es el conjunto {*rojo, verde, azul*}. Las restricciones requieren que las regiones vecinas tengan colores distintos; por ejemplo, las combinaciones aceptables para *AO* y *TN* son los pares

$$\{(rojo,verde), (rojo,azul), (verde, rojo), (verde,azul), (azul,rojo), (azul,verde)\}$$

(La restricción puede también representarse más sucintamente como la desigualdad $AO \neq TN$, a condición de que el algoritmo de satisfacción de restricciones tenga algún modo de evaluar tales expresiones.) Hay muchas soluciones posibles, como

$$\{AO = rojo, TN = verde, Q = rojo, NGS = verde, V = rojo, AS = azul, T = rojo\}$$

GRAFO DE RESTRICCIONES

Es bueno visualizar un PSR como un **grafo de restricciones**, como el que se muestra en la Figura 5.1(b). Los nodos del grafo corresponden a variables del problema y los arcos corresponden a restricciones.

Tratar un problema como un PSR confiere varias ventajas importantes. Como la representación del estado en un PSR se ajusta a un modelo estándar (es decir, un conjunto de variables con valores asignados) la función sucesor y el test objetivo pueden escribirse de un modo genérico para que se aplique a todo PSR. Además, podemos desarrollar heurísticas eficaces y genéricas que no requieran ninguna información adicional ni experta del dominio específico. Finalmente, la estructura del grafo de las restricciones puede usarse para simplificar el proceso de solución, en algunos casos produciendo una reducción exponencial de la complejidad. La representación PSR es la primera, y más simple, de una serie de esquemas de representación que serán desarrollados a través de los capítulos del libro.

Es bastante fácil ver que a un PSR se le puede dar una **formulación incremental** como en un problema de búsqueda estándar:

- **Estado inicial:** la asignación vacía { }, en la que todas las variables no están asignadas.
- **Función de sucesor:** un valor se puede asignar a cualquier variable no asignada, a condición de que no suponga ningún conflicto con variables antes asignadas.
- **Test objetivo:** la asignación actual es completa.
- **Costo del camino:** un coste constante (por ejemplo, 1) para cada paso.

Cada solución debe ser una asignación completa y por lo tanto aparecen a profundidad *n* si hay *n* variables. Además, el árbol de búsqueda se extiende sólo a profundidad *n*. Por

Figura 5.1 (a) Los estados y territorios principales de Australia. Colorear este mapa puede verse como un problema de satisfacción de restricciones. El objetivo es asignar colores a cada región de modo que ninguna de las regiones vecinas tengan el mismo color. (b) El problema del coloreo del mapa representado como un grafo de restricciones.

estos motivos, los algoritmos de búsqueda primero en profundidad son populares para PSRs. (*Véase* la Sección 5.2.) También *el camino que alcanza una solución es irrelevante*. De ahí, que podemos usar también una **formulación completa de estados**, en la cual cada estado es una asignación completa que podría o no satisfacer las restricciones. Los métodos de búsqueda local trabajan bien para esta formulación. (*Véase* la Sección 5.3.)

DOMINIOS FINITOS

La clase más simple de PSR implica variables **discretas** y **dominios finitos**. Los problemas de coloreo del mapa son de esta clase. El problema de las 8-reinas descrito en el Capítulo 3 puede también verse como un PSR con dominio finito, donde las variables $Q_1, Q_2 \ldots, Q_8$ son las posiciones de cada reina en las columnas 1…, 8 y cada variable tiene el dominio $\{1, 2, 3, 4, 5, 6, 7, 8\}$. Si el tamaño máximo del dominio de cualquier variable, en un PSR, es d, entonces el número de posibles asignaciones completas es $O(d^n)$, es decir, exponencial en el número de variables. Los PSR con dominio finito incluyen a los **PSRs booleanos**, cuyas variables pueden ser *verdaderas* o *falsas*. Los PSRs booleanos incluyen como casos especiales algunos problemas NP-completos, como 3SAT. (*Véase* el Capítulo 7.) En el caso peor, por lo tanto, no podemos esperar resolver los PSRs con dominios finitos en menos de un tiempo exponencial. En la mayoría de las aplicaciones prácticas, sin embargo, los algoritmos para PSR de uso general pueden resolver problemas de *órdenes de magnitud* más grande que los resolubles con los algoritmos de búsqueda de uso general que vimos en el Capítulo 3.

PSRs BOOLEANOS

DOMINIOS INFINITOS

Las variables discretas pueden tener también **dominios infinitos** (por ejemplo, el conjunto de números enteros o de cadenas). Por ejemplo, cuando programamos trabajos de la construcción en un calendario, la fecha de comienzo de cada trabajo es una variable y los valores posibles son números enteros de días desde la fecha actual. Con dominios infinitos, no es posible describir restricciones enumerando todas las combinaciones permitidas de valores. En cambio, se debe utilizar un **lenguaje de restricción**. Por ejemplo,

LENGUAJE DE RESTRICCIÓN

si $Trabajo_1$, que utiliza cinco días, debe preceder a $Trabajo_3$, entonces necesitaríamos un lenguaje de restricción de desigualdades algebraicas como $ComienzoTrabajo_1 + 5 \leq Comienzo\ Trabajo_3$. Tampoco es posible resolver tales restricciones enumerando todas las asignaciones posibles, porque hay infinitas. Existen algoritmos solución especiales (de los que no hablaremos aquí) para **restricciones lineales** sobre variables enteras (es decir, restricciones, como la anterior, en la que cada variable aparece de forma lineal). Puede demostrarse que no existe un algoritmo para resolver **restricciones no lineales** generales sobre variables enteras. En algunos casos, podemos reducir los problemas de restricciones enteras a problemas de dominio finito simplemente acotando los valores de todas las variables. Por ejemplo, en un problema de programación, podemos poner una cota superior igual a la longitud total de todos los trabajos a programar.

RESTRICCIONES LINEALES

RESTRICCIONES NO LINEALES

Los problemas de satisfacción de restricciones con **dominios continuos** son muy comunes en el mundo real y son ampliamente estudiados en el campo de la investigación operativa. Por ejemplo, la programación de experimentos sobre el Telescopio Hubble requiere el cronometraje muy preciso de las observaciones; al comienzo y al final de cada observación y la maniobra son variables continuas que deben obedecer a una variedad de restricciones astronómicas, prioritarias y potentes. La categoría más conocida de PSRs en dominios continuos son los problemas de **programación lineal**, en donde las restricciones deben ser desigualdades lineales que forman una región *convexa*. Los problemas de programación lineal pueden resolverse en tiempo polinomial en el número de variables. Los problemas con tipos diferentes de restricciones y funciones objetivo también se han estudiado: programación cuadrática, programación cónica de segundo orden, etcétera.

DOMINIOS CONTINUOS

PROGRAMACIÓN LINEAL

Además del examen de los tipos de variables que pueden aparecer en los PSRs, es útil ver los tipos de restricciones. El tipo más simple es la **restricción unaria**, que restringe los valores de una sola variable. Por ejemplo, podría ser el caso en los que a los australianos del Sur les disgustara el color *verde*. Cada restricción unaria puede eliminarse simplemente con el preproceso del dominio de la variable quitando cualquier valor que viole la restricción. Una **restricción binaria** relaciona dos variables. Por ejemplo, AS $\neq$ NGS es una restricción binaria. Un PSR binario es un problema con restricciones sólo binarias; y puede representarse como un grafo de restricciones, como en la Figura 5.1(b).

RESTRICCIÓN UNARIA

RESTRICCIÓN BINARIA

Las restricciones de orden alto implican tres o más variables. Un ejemplo familiar es el que proporcionan los puzles **cripto-aritméticos**. (*Véase* la Figura 5.2(a).) Es habitual insistir que cada letra en un puzle cripto-aritmético represente un dígito diferente. Para el caso de la Figura 5.2(a), éste sería representado como la restricción de seis variables *TodasDif* (*F, T, U, W, R, O*). O bien, puede representarse por una colección de restricciones binarias como $F \neq T$. Las restricciones añadidas sobre las cuatro columnas del puzle también implican varias variables y pueden escribirse como

CRIPTO-ARITMÉTICO

$$
\begin{aligned}
&O + O = R + 10 \cdot X_1 \\
&X_1 + W + W = U + 10 \cdot X_2 \\
&X_2 + T + T = O + 10 \cdot X_3 \\
&X_3 = F
\end{aligned}
$$

donde X_1, X_2, y X_3 son **variables auxiliares** que representan el dígito (0 o 1) transferido a la siguiente columna. Las restricciones de orden alto pueden representarse en un **hiper-grafo de restricciones**, como el de la Figura 5.2(b). El lector habrá notado que

VARIABLES AUXILIARES

HIPER-GRAFO DE RESTRICCIONES

Figura 5.2 (a) Un problema cripto-aritmético. Cada letra significa un dígito distinto; el objetivo es encontrar una sustitución de las letras por dígitos tal que la suma que resulta es aritméticamente correcta, con la restricción añadida de que no se permite la administración de ceros. (b) El hiper-grafo de restricciones para el problema cripto-aritmético muestra la restricción *TodasDif* así como las restricciones añadidas en las columnas. Cada restricción es una caja cuadrada relacionada con las variables que restringe.

la restricción *TodasDif* puede dividirse en restricciones binarias ($F \neq T$, $F \neq U$, etcétera). De hecho, como en el Ejercicio 5.11, se pide demostrar que la restricción de dominio finito y orden alto puede reducirse a un conjunto de restricciones binarias si se introducen suficientes variables auxiliares. A causa de esto, en este capítulo trataremos sólo restricciones binarias.

Las restricciones que hemos descrito hasta ahora han sido todas restricciones **absolutas**, la violación de las cuales excluye una solución potencial. Muchos PSRs del mundo real incluyen restricciones de **preferencia** que indican qué soluciones son preferidas. Por ejemplo, en un problema de horario de la universidad, el profesor X quizá prefiera enseñar por la mañana mientras que un profesor Y prefiere enseñar por la tarde. Un horario que tenga al profesor X enseñando a las dos de la tarde sería una solución (a menos que el profesor X resulte ser el director del departamento), pero no sería un óptimo. Las restricciones de preferencia pueden a menudo codificarse como costos sobre las asignaciones de variables individuales (por ejemplo, asignar un hueco por la tarde para el profesor X cuesta dos puntos contra la función objetivo total, mientras que un hueco de mañana cuesta uno). Con esta formulación, los PSRs con preferencias pueden resolverse utilizando métodos de búsqueda de optimización, basados en caminos o locales. No hablamos de tales PSRs fuera de este capítulo, pero sí proporcionamos algunas líneas de trabajo en la sección de notas bibliográficas.

PREFERENCIA

5.2 Búsqueda con vuelta atrás para PSR

La sección anterior dio una formulación de PSRs como problemas de búsqueda. Usando esta formulación, cualquiera de los algoritmos de búsqueda de los Capítulos 3 y 4

pueden resolver los PSRs. Suponga que aplicamos la búsqueda primero en anchura a la formulación del PSR genérico de la sección anterior. Rápidamente notamos algo terrible: el factor de ramificación en el nivel superior es de nd, porque cualquiera de los d valores se puede asignar a cualquiera de las n variables. En el siguiente nivel, el factor de ramificación es $(n\text{-}1)d$, etcétera para los n niveles. ¡Generamos un árbol con $n! \cdot d^n$ hojas, aunque haya sólo d^n asignaciones posibles completas!

CONMUTATIVIDAD

Nuestra formulación del problema, aparentemente razonable pero ingenua, no ha hecho caso de una propiedad crucial común en todos los PSRs: la **conmutatividad**. Un problema es conmutativo si el orden de aplicación de cualquier conjunto de acciones no tiene ningún efecto sobre el resultado. Éste es el caso de los PSRs porque, asignando valores a variables, alcanzamos la misma asignación parcial sin tener en cuenta el orden. Por lo tanto, *todos los algoritmos de búsqueda para el PSR generan los sucesores considerando asignaciones posibles para sólo una* variable en cada nodo del árbol de búsqueda. Por ejemplo, en el nodo raíz de un árbol de búsqueda para colorear el mapa de Australia, podríamos tener una opción entre $AS = rojo$, $AS = verde$, y $AS = azul$, pero nunca elegiríamos entre $AS = rojo$ y $AO = azul$. Con esta restricción, el número de hojas es d^n, como era de esperar.

BÚSQUEDA CON VUELTA ATRÁS

El término **búsqueda con vuelta atrás** se utiliza para la búsqueda primero en profundidad que elige valores para una variable a la vez y vuelve atrás cuando una variable no tiene ningún valor legal para asignarle. En la Figura 5.3 se muestra este algoritmo. Notemos que usa, en efecto, el método uno a la vez de la generación de sucesor incremental descrita en la página 86. También, extiende la asignación actual para generar un sucesor, más que volver a copiarlo. Como la representación de los PSRs está estandartizada, no hay ninguna necesidad de proporcionar a la BÚSQUEDA-CON-VUELTA-ATRÁS un estado inicial del dominio específico, una función sucesor, o un test del objetivo. En la Figura 5.4 se muestra parte del árbol de búsqueda para el problema de Australia, en donde hemos asignado variables en el orden AO, TN, Q...

```
función BÚSQUEDA-CON-VUELTA-ATRÁS(psr) devuelve una solución, o fallo
  devolver VUELTA-ATRÁS-RECURSIVA({ }, psr)

función VUELTA-ATRÁS-RECURSIVA(asignación, psr) devuelve una solución, o fallo
  si asignación es completa entonces devolver asignación
  var ← SELECCIONA-VARIABLE-NOASIGNADA(VARIABLES[psr], asignación, psr)
  para cada valor en ORDEN-VALORES-DOMINIO(var, asignación, psr) hacer
  si valor es consistente con asignación de acuerdo a las RESTRICCIONES[psr] entonces
    añadir {var = valor} a asignación
    resultado ← VUELTA-ATRÁS-RECURSIVA(asignación, psr)
    si resultado ≠ fallo entonces devolver resultado
    borrar {var = valor} de asignación
  devolver fallo
```

Figura 5.3 Un algoritmo simple de vuelta atrás para problemas de satisfacción de restricciones. El algoritmo se modela sobre la búsqueda primero en profundidad recursivo del Capítulo 3. Las funciones SELECCIONA-VARIABLE-NOASIGNADA y ORDEN-VALORES-DOMINIO pueden utilizarse para implementar las heurísticas de propósito general discutidas en el texto.

Figura 5.4 Parte del árbol de búsqueda generado por vuelta atrás para el problema que colorea el mapa de la Figura 5.1.

La vuelta atrás sencilla es un algoritmo sin información en la terminología del Capítulo 3, así que no esperamos que sea muy eficaz para problemas grandes. En la primera columna de la Figura 5.5 se muestran los resultados para algunos problemas y se confirman nuestras expectativas.

En el Capítulo 4 remediamos el funcionamiento pobre de los algoritmos de búsqueda sin información suministrándoles funciones heurísticas específicas del dominio obtenidas de nuestro conocimiento del problema. Resulta que podemos resolver los PSRs de

Problema	Vuelta atrás	VA + MVR	Comprobación hacia delante	CD + MVR	Mínimo conflicto
EE.UU.	(> 1.000K)	(> 1.000K)	2K	60	64
n-reinas	(> 40.000K)	13.500K	(> 40.000K)	817K	4K
Zebra	3.859K	1K	35K	0,5K	2K
Aleatorio 1	415K	3K	26K	2K	
Aleatorio 2	942K	27K	77K	15K	

Figura 5.5 Comparación de varios algoritmos de PSR sobre varios problemas. Los algoritmos, de izquierda a derecha, son vuelta atrás simple, vuelta atrás con la heurística MVR, comprobación hacia delante, comprobación hacia delante con MVR y búsqueda local de conflictos mínimos. En cada celda está el número medio de comprobaciones consistentes (sobre cinco ejecuciones) requerido para resolver el problema; notemos que todas las entradas excepto las dos de la parte superior derecha están en miles (K). Los números en paréntesis significan que no se ha encontrado ninguna respuesta en el número asignado de comprobaciones. El primer problema es colorear, con cuatro colores, los 50 estados de los Estados Unidos de América. Los problemas restantes se han tomado de Bacchus y van Run (1995), Tabla 1. El segundo problema cuenta el número total de comprobaciones requeridas para resolver todos los problemas de *n*-reinas para *n* de dos a 50. El tercero es el «puzle Zebra», descrito en el Ejercicio 5.13. Los dos últimos son problemas artificiales aleatorios. (Mínimos conflictos no se ejecutó sobre estos.) Los resultados sugieren que la comprobación hacia delante con la heurística MVR es mejor sobre todos estos problemas que los otros algoritmos con vuelta atrás, pero no siempre es mejor que los de búsqueda local de mínimo conflicto.

manera eficiente sin tal conocimiento específico del dominio. En cambio, encontramos métodos de propósito general que proporcionan las siguientes preguntas:

1. ¿Qué variable debe asignarse después, y en qué orden deberían intentarse sus valores?
2. ¿Cuáles son las implicaciones de las asignaciones de las variables actuales para las otras variables no asignadas?
3. Cuándo un camino falla, es decir, un estado alcanzado en el que una variable no tiene ningún valor legal, ¿puede la búsqueda evitar repetir este fracaso en caminos siguientes?

Las subsecciones siguientes contestan a cada una de estas preguntas.

Variable y ordenamiento de valor

El algoritmo con vuelta atrás contiene la línea

var ← SELECCIONA-VARIABLE-NOASIGNADA(VARIABLES[*psr*], *asignación, psr*)

Por defecto, SELECCIONA-VARIABLE-NOASIGNADA simplemente selecciona la siguiente variable no asignada en el orden dado por la lista VARIABLES[*psr*]. Esta variable estática, rara vez ordenada, da como resultado una búsqueda más eficiente. Por ejemplo, después de las asignaciones para $AO = rojo$ y $T = verde$, hay un sólo valor posible para *AS*, entonces tiene sentido asignar $AS = azul$ a continuación más que asignar un valor a *Q*. De hecho, después de asignar *AS*, las opciones para *Q, NGS*, y *V* están forzadas. Esta idea intuitiva (escoger la variable con menos valores «legales») se llama heurística de **mínimos valores restantes** (MVR). También llamada heurística «variable más restringida» o «primero en fallar», este último es porque escoge una variable que con mayor probabilidad causará pronto un fracaso, con lo cual podamos el árbol de búsqueda. Si hay una variable *X* con cero valores legales restantes, la heurística MVR seleccionará *X* y el fallo será descubierto inmediatamente (evitando búsquedas inútiles por otras variables que siempre fallarán cuando se seleccione *X* finalmente). La segunda columna de la Figura 5.5, etiquetada con VA + MVR, muestra el funcionamiento de esta heurística. El funcionamiento es de tres a 3.000 veces mejor que la vuelta atrás simple, según el problema. Notemos que nuestra medida de rendimiento no hace caso del coste suplementario de calcular los valores heurísticos; la siguiente subsección describe un método que maneja este coste.

MÍNIMOS VALORES RESTANTES

La heurística MVR no ayuda en absoluto en la elección de la primera región a colorear en Australia, porque al principio cada región tiene tres colores legales. En este caso, el **grado heurístico** es más práctico. Intenta reducir el factor de ramificación sobre futuras opciones seleccionando la variable, entre las variables no asignadas, que esté implicada en el mayor número de restricciones. En la Figura 5.1, *AS* es la variable con el grado más alto, cinco; las otras variables tienen grado dos o tres, excepto *T*, que tiene cero. De hecho, una vez elegida *AS*, aplicando el grado heurístico que resuelve el problema sin pasos en falso puede elegir cualquier color consistente en cada punto seleccionado y todavía llegar a una solución sin vuelta atrás. La heurística del mínimo de valores restantes es por lo general una guía más poderosa, pero el grado heurístico puede ser útil como un desempate.

GRADO HEURÍSTICO

VALOR MENOS RESTRINGIDO

Una vez que se selecciona una variable, el algoritmo debe decidir el orden para examinar sus valores. Para esto, la heurística del **valor menos restringido** puede ser eficaz en algunos casos. Se prefiere el valor que excluye las pocas opciones de las variables vecinas en el grafo de restricciones. Por ejemplo, supongamos que en la Figura 5.1 hemos generado la asignación parcial con *AO* = *roja* y *TN* = *verde*, y que nuestra siguiente opción es para *Q*. Azul sería una opción mala, porque elimina el último valor legal del vecino de *Q*, *SA*. La heurística del valor menos restringido, por lo tanto, prefiere rojo al azul. En general, la heurística trata de dejar la flexibilidad máxima de las asignaciones de las variables siguientes. Desde luego, si tratamos de encontrar todas las soluciones a un problema, no sólo la primera, entonces el orden no importa porque tenemos que considerar cada valor de todos modos. Ocurre lo mismo si no hay soluciones al problema.

Propagación de la información a través de las restricciones

Hasta ahora nuestro algoritmo de búsqueda considera las restricciones sobre una variable sólo cuando la variable es elegida por SELECCIONA-VARIABLE-NOASIGNADA. Pero mirando algunas restricciones antes en la búsqueda, o incluso antes de que haya comenzado la búsqueda, podemos reducir drásticamente el espacio de ésta.

Comprobación hacia delante

COMPROBACIÓN HACIA DELANTE

Otra manera para usar mejor las restricciones durante la búsqueda se llama **comprobación hacia delante**. Siempre que se asigne una variable *X*, el proceso de comprobación hacia delante mira cada variable no asignada *Y* que esté relacionada con *X* por una restricción y suprime del dominio de *Y* cualquier valor que sea inconsistente con el valor elegido para *X*. La Figura 5.6 muestra el progreso de una búsqueda, que colorea un mapa, con la comprobación hacia delante. Hay dos puntos importantes que debemos destacar sobre este ejemplo. Primero, notemos que después de asignar *AO* = *rojo* y *Q* = *verde*, los dominios de *TN* y *AS* se reducen a un solo valor; hemos eliminado las ramificaciones de estas variables totalmente propagando la información de *AO* y *Q*. La

	AO	*TN*	*Q*	*NGS*	*V*	*AS*	*T*
Dominios iniciales	R V A	R V A	R V A	R V A	R V A	R V A	R V A
Después de *AO*=*rojo*	Ⓡ	V A	R V A	R V A	R V A	V A	R V A
Después de *Q*=*verde*	Ⓡ	A	Ⓥ	R A	R V A	A	R V A
Después de *V*=*azul*	Ⓡ	A	Ⓥ	R	Ⓐ		R V A

Figura 5.6 Progreso de una búsqueda, que colorea un mapa, con comprobación hacia delante. *AO* = *rojo* se asigna primero; entonces la comprobación hacia delante suprime *rojo* de los dominios de las variables vecinas *TN* y *AS*. Después *Q* = *verde*; *verde* se suprime de los dominios *TN*, *AS*, y *NGS*. Después *V* = *azul*; *azul* se suprime del dominio de *NGS* y *AS*, y se obtiene AS sin valores legales.

heurística MVR, la cual es una compañera obvia para la comprobación hacia delante, seleccionaría automáticamente *AS* y *TN* después. (En efecto, podemos ver la comprobación hacia delante como un modo eficiente de incrementar el cálculo de la información que necesita la heurística MVR para hacer su trabajo.) Un segundo punto a tener en cuenta es que, después de *V = azul*, el dominio de *SA* está vacío. Por eso, la comprobación hacia delante ha descubierto que la asignación parcial {*AO = rojo, Q = verde, V = azul*} es inconsistente con las restricciones del problema, y el algoritmo volverá atrás inmediatamente.

Propagación de restricciones

Aunque la comprobación hacia delante descubre muchas inconsistencias, no descubre todas. Por ejemplo, considere la tercera fila de la Figura 5.6. Muestra que cuando *AO* es *rojo* y *Q* es *verde*, tanto *TN* como *AS* están obligadas a ser azules. Pero son adyacentes y por tanto no pueden tener el mismo valor. La comprobación hacia delante no la descubre como una inconsistencia, porque no mira lo bastante lejos. La **propagación de restricciones** es el término general para la propagación de las implicaciones de una restricción sobre una variable en las otras variables; en este caso necesitamos propagar desde *AO* y *Q* a *TN* y *AS*, (como se hizo con la comprobación hacia delante) y luego a la restricción entre *TN* y *AS* para descubrir la inconsistencia. Y queremos hacer esto rápido: no es nada bueno reducir la cantidad de búsqueda si gastamos más tiempo propagando restricciones que el que hubiéramos gastado haciendo una búsqueda simple.

PROPAGACIÓN DE RESTRICCIONES

ARCO CONSISTENTE

La idea del **arco consistente** proporciona un método rápido de propagación de restricciones que es considerablemente más potente que la comprobación hacia delante. Aquí, «el arco» se refiere a un arco dirigido en el grafo de restricciones, como el arco de *AS* a *NGS*. Considerando los dominios actuales de *AS* y *NGS*, el arco es consistente si, para *todo* valor *x* de *AS*, hay *algún* valor *y* de *NGS* que es consistente con *x*. En la tercera fila de la Figura 5.6, los dominios actuales de *AS* y *NGS* son {*azul*} y {*rojo, azul*}, respectivamente. Para *AS = azul*, hay una asignación consistente para *NGS*, a saber, *NGS = roja*; por lo tanto, el arco de *AS* a *NGS* es consistente. Por otra parte, el arco inverso desde *NGS* a *AS* no es consistente: para la asignación *NGS = azul*, no hay ninguna asignación consistente para *AS*. El arco puede hacerse consistente suprimiendo el valor *azul* del dominio de *NGS*.

Podemos aplicar también el arco consistente al arco de *AS* a *TN* en la misma etapa del proceso de búsqueda. La tercera fila de la tabla en la Figura 5.6 muestra que ambas variables tienen el dominio {*azul*}. El resultado es que *azul* debe suprimirse del dominio de *AS*, dejando el dominio vacío. Así, la aplicación del arco consistente ha causado la detección temprana de una inconsistencia que no es descubierta por la comprobación hacia delante pura.

La comprobación de la consistencia del arco puede aplicarse como un paso de preproceso antes de comenzar el proceso de búsqueda, o como un paso de propagación (como la comprobación hacia delante) después de cada asignación durante la búsqueda (a veces se llama a este último algoritmo MCA, *Mantenimiento de la Consistencia del Arco*). En uno u otro caso, el proceso debe aplicarse *repetidamente* hasta que no permanezcan

más inconsistencias. Esto es porque, siempre que un valor se suprime del dominio de alguna variable para quitar una inconsistencia de un arco, una nueva inconsistencia de arco podría surgir en arcos que señalan a aquella variable. El algoritmo completo para la consistencia de arco, AC-3, utiliza una cola para guardar los arcos que tienen que comprobarse (*véase* la Figura 5.7). Cada arco (X_i, X_j) se quita sucesivamente de la agenda y se comprueba; si cualquier valor requiere ser suprimido del dominio de X_i, entonces cada arco (X_k, X_i) señalando a X_i debe ser reinsertado sobre la cola para su comprobación. La complejidad de la comprobación de consistencia de arco puede analizarse como sigue: un PSR binario tiene a lo más $O(n^2)$ arcos; cada arco (X_k, X_i) puede insertarse en la agenda sólo d veces, porque X_i tiene a lo más d valores para suprimir; la comprobación de la consistencia de un arco puede hacerse en $O(d^2)$ veces; entonces el tiempo total, en el caso peor, es $O(n^2d^3)$. Aunque sea considerablemente más costoso que la comprobación hacia delante, el coste suplementario por lo general vale la pena[1].

Como los PSRs incluyen a 3SAT como un caso especial, no esperamos encontrar un algoritmo de tiempo polinomial que puede decidir si un PSR es consistente. De ahí, deducimos que la consistencia de arco no revela todas las inconsistencias posibles. Por ejemplo, en la Figura 5.1, la asignación parcial $\{AO = rojo, NGS = rojo\}$ es inconsistente, pero AC-3 no la encuentra. Las formas más potentes de la propagación pueden definirse utilizando la noción llamada k-**consistencia**. Un PSR es k-consistente si, para cualquier conjunto de $k - 1$ variables y para cualquier asignación consistente a esas variables, siempre se puede asignar un valor consistente a cualquier k-ésima variable.

***K*-CONSISTENCIA**

```
función AC-3(psr) devuelve el PSR, posiblemente con dominio reducido
   entradas: psr, un PSR binario con variables {X1, X2, …, Xn}
   variables locales: cola, una cola de arcos, inicialmente todos los arcos del psr

   mientras cola es no vacía hacer
      (Xi, Xj) ← BORRAR-PRIMERO(cola)
      si BORRAR-VALORES-INCONSISTENTES(Xi, Xj) entonces
         para cada Xk en VECINOS[Xi] hacer
            añadir(Xk, Xi) a la cola
```

```
función BORRAR-VALORES-INCONSISTENTES(Xi, Xj) devuelve verdadero si y sólo si hemos
borrado un valor
   borrado ← falso
   para cada x en DOMINIO[Xi] hacer
      si no hay un y en DOMINIO[Xj] que permita a (x,y) satisfacer la restricción entre Xi y Xj
         entonces borrar x de DOMINIO[Xi]; borrado × verdadero
   devolver borrado
```

Figura 5.7 El algoritmo AC-3 de la consistencia del arco. Después de aplicar AC-3, cada arco es arco-consistente, o alguna variable tiene un dominio vacío, indicando que el PSR no puede hacerse arco-consistente (y así el PSR no puede resolverse). El nombre «AC-3» fue usado por el inventor del algoritmo (Mackworth, 1977) porque era la tercera versión desarrollada en el artículo.

[1] El algoritmo AC-4, debido a Mohr y Henderson (1986), se ejecuta en $O(n^2d^2)$. *Véase* el Ejercicio 5.10.

CONSISTENCIA DE NODO

CONSISTENCIA DE CAMINO

FUERTEMENTE *K*-CONSISTENTE

Por ejemplo, la 1-consistencia significa que cada variable individual, por sí mismo, es consistente; también llamado **consistencia de nodo**. La 2-consistencia es lo mismo que la consistencia de arco. La 3-consistencia significa que cualquier par de variables adyacentes pueden siempre extenderse a una tercera variable vecina; también llamada **consistencia de camino**.

Un grafo es **fuertemente** k-**consistente** si es k-consistente y también $(k - 1)$-consistente, $(k - 2)$-consistente..., hasta 1-consistente. Ahora supongamos que tenemos un PSR con n nodos y lo hacemos fuertemente n-consistente (es decir, fuertemente k-consistente para $k = n$). Podemos resolver el problema sin vuelta atrás. Primero, elegimos un valor consistente para X_1. Entonces tenemos garantizado el poder elegir un valor para X_2 porque el grafo es 2-consistente, para X_3 porque es 3-consistente, etcétera. Para cada variable X_i, tenemos sólo que averiguar los valores de d, en el dominio, para encontrar un valor consistente con $X_1 ..., X_{i-1}$. Tenemos garantizado encontrar una solución en $O(nd)$. Desde luego, no tenemos ningún tiempo libre: cualquier algoritmo para establecer la n-consistencia debe llevar un tiempo exponencial en n, en el caso peor.

Hay una amplia diferencia entre consistencia de arco y n-consistencia: ejecutar controles de consistencia fuerte llevará más tiempo, pero tendrá un efecto mayor en la reducción del factor de ramificación y en descubrir asignaciones parciales inconsistentes. Es posible calcular el valor k más pequeño tal que al ejecutar la k-consistencia asegura que el problema puede resolverse sin volver atrás (*véase* la Sección 5.4), pero a menudo es poco práctico. En la práctica, la determinación del nivel apropiado de comprobación de la consistencia es sobre todo una ciencia empírica.

Manejo de restricciones especiales

Cierto tipo de restricciones aparecen con frecuencia en problemas reales y pueden manejarse utilizando algoritmos de propósito especial más eficientes que los métodos de propósito general descritos hasta ahora. Por ejemplo, la restricción *Todasdif* dice que todas las variables implicadas deben tener valores distintos (como en el problema criptoaritmético). Una forma simple de detección de inconsistencias para la restricción *Todasdif* trabaja como sigue: si hay m variables implicadas en la restricción, y si tienen n valores posibles distintos, y $m > n$, entonces la restricción no puede satisfacerse.

Esto nos lleva al algoritmo simple siguiente: primero, quite cualquier variable en la restricción que tenga un dominio con una sola posibilidad, y suprima el valor de esa variable de los dominios de las variables restantes. Repetir mientras existan variables con una sola posibilidad. Si en algún momento se produce un dominio vacío o hay más variables que valores en el dominio, entonces se ha descubierto una inconsistencia.

Podemos usar este método para detectar la inconsistencia en la asignación parcial $\{AO = rojo, NGS = rojo\}$ para la Figura 5.1. Notemos que las variables SA, TN y Q están efectivamente relacionadas por una restricción *Todasdif* porque cada par debe ser de un color diferente. Después de aplicar AC-3 con la asignación parcial, el dominio de cada variable se reduce a $\{verde, azul\}$. Es decir, tenemos tres variables y sólo dos colores, entonces se viola la restricción *Todasdif*. Así, un procedimiento simple de consistencia para una restricción de orden alto es a veces más eficaz que la aplicación de la consistencia de arco a un conjunto equivalente de restricciones binarias.

RESTRICCIÓN DE RECURSOS

Quizá la restricción de orden alto más importante es la **restricción de recursos**, a veces llamada restricción *como-máximo*. Por ejemplo, $PA_1 \ldots, PA_4$ denotan los números de personas asignadas a cada una de las cuatro tareas. La restricción que no asigna más de 10 personas en total, se escribe *como-máximo*$(10, PA_1, PA_2, PA_3, PA_4)$. Se puede descubrir una inconsistencia simplemente comprobando la suma de los valores mínimos de los dominios actuales; por ejemplo, si cada variable tiene el dominio $\{3, 4, 5, 6\}$, la restricción *como-máximo* no puede satisfacerse. También podemos hacer cumplir la consistencia suprimiendo el valor máximo de cualquier dominio si no es consistente con los valores mínimos de los otros dominios. Así, si cada variable, en nuestro ejemplo, tiene el dominio $\{2, 3, 4, 5, 6\}$, los valores 5 y 6 pueden suprimirse de cada dominio.

Para problemas grandes de recursos limitados con valores enteros (como son los problemas logísticos que implican el movimiento de miles de personas en cientos de vehículos) no es, por lo general, posible representar el dominio de cada variable como un conjunto grande de enteros y gradualmente reducir ese conjunto por los métodos de comprobación de la consistencia. En cambio, los dominios se representan por límites superiores e inferiores y son manejados por la propagación de límites. Por ejemplo, supongamos que hay dos vuelos, 271 y 272, para los cuales los aviones tienen capacidades de 156 y 385, respectivamente. Los dominios iniciales para el número de pasajeros sobre cada vuelo son

$$Vuelo271 \in [0,165] \quad \text{y} \quad Vuelo272 \in [0,385]$$

Supongamos ahora que tenemos la restricción adicional que los dos vuelos juntos deben llevar a las 420 personas: $Vuelo271 + Vuelo272 \in [420,420]$. Propagando los límites de las restricciones, reducimos los dominios a

$$Vuelo271 \in [35,165] \quad \text{y} \quad Vuelo272 \in [255,385]$$

PROPAGACIÓN DE LÍMITES

Decimos que un PSR es consistente-acotado si para cada variable X, y tanto para los valores de las cotas inferior y superior de X, existe algún valor de Y que satisface la restricción entre X e Y, para cada variable Y. Esta clase de **propagación de límites** se utiliza, ampliamente, en problemas restringidos prácticos.

Vuelta atrás inteligente: mirando hacia atrás

VUELTA ATRÁS CRONOLÓGICA

El algoritmo de BÚSQUEDA-CON-VUELTA-ATRÁS de la Figura 5.3 tiene una política muy simple para saber qué hacer cuando falla una rama de búsqueda: hacia atrás hasta la variable anterior e intentar un valor diferente para ella. Se llama **vuelta atrás cronológica**, porque se visita de nuevo el punto de decisión *más reciente*. En esta subsección, veremos que hay muchos caminos mejores.

Veamos lo que ocurre cuando aplicamos la vuelta atrás simple de la Figura 5.1 con una variable fija que ordena Q, NGS, V, T, AS, AO, TN. Supongamos que hemos generado la asignación parcial $\{Q = rojo, NGS = verde, V = azul, T = rojo\}$. Cuando intentamos la siguiente variable, AS, vemos que cada valor viola una restricción. ¡Volvemos hacia atrás hasta T e intentamos un nuevo color para Tasmania! Obviamente esto es inútil (el nuevo color de Tasmania no puede resolver el problema con Australia del Sur).

CONJUNTO CONFLICTO

SALTO-ATRÁS

Una aproximación más inteligente a la vuelta atrás es ir hacia atrás hasta el conjunto de variables que *causaron el fracaso*. A este conjunto se le llama **conjunto conflicto**; aquí, el conjunto conflicto para *AS* es $\{Q, NGS, V\}$. En general, el conjunto conflicto para la variable *X* es el conjunto de variables previamente asignadas que están relacionadas con *X* por las restricciones. El método **salto-atrás** retrocede a la variable *más reciente* en el conjunto conflicto; en este caso, el salto-atrás debería saltar sobre Tasmania e intentar un nuevo valor para *V*. Esto se implementa fácilmente modificando la BÚSQUEDA-CON-VUELTA-ATRÁS de modo que acumule el conjunto conflicto mientras comprueba un valor legal para asignar. Si no se encuentra un valor legal, debería devolver el elemento más reciente del conjunto conflicto con el indicador de fracaso.

El lector habrá notado que la comprobación hacia delante puede suministrar el conjunto conflicto sin trabajo suplementario: siempre que la comprobación hacia delante, basada en una asignación a *X*, suprima un valor del dominio de *Y*, deberíamos añadir *X* al conjunto conflicto de *Y*. También, siempre que se suprima el último valor del dominio de *Y*, las variables en el conjunto conflicto de *Y* se añaden al conjunto conflicto de *X*. Entonces, cuando llegamos a *Y*, sabemos inmediatamente dónde volver atrás si es necesario.

El lector habrá notado algo raro: el salto-atrás ocurre cuando cada valor de un dominio está en conflicto con la asignación actual; ¡pero la comprobación hacia delante descubre este acontecimiento y previene la búsqueda de alcanzar alguna vez tal nodo! De hecho, puede demostrarse que *cada rama podada por el salto-atrás también se poda por la comprobación hacia delante*. De ahí, el salto-atrás simple es redundante con una búsqueda de comprobación hacia delante o, en efecto, en una búsqueda que usa la comprobación de consistencia fuerte, como MCA.

A pesar de las observaciones del párrafo anterior, la idea que hay detrás del salto-atrás sigue siendo buena: retroceder basándose en los motivos de fracaso. El salto-atrás se da cuenta del fracaso cuando el dominio de una variable se hace vacío, pero en muchos casos una rama es condenada mucho antes de que esto ocurra. Considere otra vez la asignación parcial $\{AO = rojo, NGS = rojo\}$ (que, de nuestra discusión anterior, es inconsistente). Supongamos que intentamos $T = rojo$ después y luego asignamos a *TN, Q, V, AS*. Sabemos que no se puede hacer ninguna asignación para estas cuatro últimas variables, ya que finalmente nos quedamos sin valores para intentar en *TN*. Ahora, la pregunta es, ¿a dónde regresar? El salto-atrás no puede trabajar, porque *TN* tiene realmente valores consistentes con las variables precedentes adjudicadas (*TN* no tiene un conjunto conflicto completo de variables precedentes que hicieron que fallara). Sabemos, sin embargo, que las cuatro variables *TN, Q, V,* y *AS*, *juntas*, dan fallo debido a un conjunto de variables precedentes, que directamente entran en conflicto con las cuatro. Esto conduce a una noción más profunda del conjunto conflicto para una variable como *TN*: es ese conjunto de variables precedentes el que causa que *TN*, junto con cualquier variable siguiente, no tenga ninguna solución consistente. En este caso, el conjunto es *AO* y *NGS*, así que el algoritmo debería regresar a *NGS* y saltarse Tasmania. Al algoritmo de salto–atrás que utiliza los conjuntos conflicto definidos de esta manera se le llama **salto-atrás dirigido-por-conflicto**.

SALTO-ATRÁS DIRIGIDO POR CONFLICTO

Debemos explicar ahora cómo calculamos estos nuevos conjuntos conflictos. El método, de hecho, es muy simple. El fracaso «terminal» de una rama de búsqueda siempre

ocurre porque el dominio de una variable se hace vacío; esa variable tiene un conjunto conflicto estándar. En nuestro ejemplo, *AS* falla, y su conjunto conflicto es $\{AO, TN, Q\}$. Saltamos atrás a *Q*, y *Q absorbe* el conjunto conflicto de *AS* (menos *Q*, desde luego) en su propio conjunto conflicto directo, que es $\{TN, NGS\}$; el nuevo conjunto conflicto es $\{AO, TN, NGS\}$. Es decir no hay ninguna solución de *Q* hacia delante, dada la asignación precedente a $\{AO, TN, NGS\}$. Por lo tanto, volvemos atrás a *NT*, el más reciente de éstos. *TN* absorbe $\{AO, TN, NGS\} - \{TN\}$ en su propio conjunto conflicto directo $\{AO\}$, dando $\{AO, NGS\}$ (como en los párrafos anteriores). Ahora el algoritmo de salto atrás a *NGS*, como era de esperar. Resumiendo: sea X_j la variable actual, y sea $conf(X_j)$ su conjunto conflicto. Si para todo valor posible para X_j falla, saltamos atrás a la variable más reciente X_i, en $conf(X_j)$, y el conjunto

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_i\}$$

APRENDER LA RESTRICCIÓN

Salto-atrás dirigido-por-conflicto va hacia atrás al punto derecho en el árbol de búsqueda, pero no nos impide cometer los mismos errores en otra rama del árbol. **Aprender la restricción** realmente modifica el PSR añadiendo una nueva restricción inducida por estos conflictos.

5.3 Búsqueda local para problemas de satisfacción de restricciones

Los algoritmos de búsqueda local (*véase* la Sección 4.3) resultan ser muy eficaces en la resolución de muchos PSRs. Ellos utilizan una formulación de estados completa: el estado inicial asigna un valor a cada variable, y la función sucesor, por lo general, trabaja cambiando el valor de una variable a la vez. Por ejemplo, en el problema de las 8-reinas, el estado inicial podría ser una configuración arbitraria de ocho reinas en ocho columnas, y la función sucesor escoge a una reina y piensa en moverla a otra parte en su columna. Otra posibilidad sería comenzar con ocho reinas, una por columna, en una permutación de las ocho filas, y generaría a un sucesor tomando dos reinas que intercambian sus filas[2]. Hemos visto ya, realmente, un ejemplo de búsqueda local para resolver un PSR: la aplicación de las ascensión de colinas al problema de n-reinas (página 126). Otra es la aplicación de SAT-CAMINAR (página 251) para resolver problemas de satisfacción, un caso especial de PSRs.

MÍNIMO CONFLICTO

En la elección de un nuevo valor para una variable, la heurística más obvia debe seleccionar el valor que cause el número mínimo de conflictos con otras variables (heurística de **mínimos-conflictos**). La Figura 5.8 muestra el algoritmo y en la Figura 5.9 se muestra su aplicación a un problema de 8-reinas, cuantificada en la Figura 5.5.

Los mínimos-conflictos son sorprendentemente eficaces para muchos PSRs, en particular, cuando se ha dado un estado inicial razonable. En la última columna de la Figu-

[2] La búsqueda local puede extenderse fácilmente a PSRs con funciones objetivo. En este caso, todas las técnicas de ascensión de colinas y temple simulado pueden aplicarse para optimizar la función objetivo.

```
función MÍNIMOS-CONFLICTOS(psr,max_pasos) devuelve una solución o fallo
  variables de entrada: psr, un problema de satisfacción de restricciones
                        max_pasos, número de pasos permitidos antes de abandonar

  actual ← una asignación completa inicial para psr
  para i = 1 hasta max_pasos hacer
    si actual es una solución para psr entonces devolver actual
    var ← escoger aleatoriamente una variable conflictiva de VARIABLES[psr]
    valor ← el valor v para var que minimiza CONFLICTOS(var, v, actual, psr)
    conjunto var = valor en actual
  devolver fallo
```

Figura 5.8 El algoritmo de MÍNIMOS-CONFLICTOS para resolver PSRs por búsqueda local. El estado inicial puede elegirse al azar o por un proceso de asignación voraz que elige un valor de mínimo conflicto para cada variable a cambiar. La función CONFLICTOS cuenta el número de restricciones violadas por un valor particular, considerando el resto de la asignación actual.

Figura 5.9 Una solución de dos pasos para un problema de 8-reinas usando mínimos-conflictos. En cada etapa, se elige una reina para la reasignación en su columna. El número de conflictos (en este caso, el número de reinas atacadas) se muestra en cada cuadrado. El algoritmo mueve a la reina al cuadrado de mínimo conflicto, deshaciendo los empates de manera aleatoria.

ra 5.5 se muestra su funcionamiento. Extraordinariamente, sobre el problema de las *n*-reinas, si no cuenta la colocación inicial de las reinas, el tiempo de ejecución de mínimos-conflictos es más o menos *independiente del tamaño del problema*. Resuelve hasta el problema de un *millón* de reinas en un promedio de 50 pasos (después de la asignación inicial). Esta observación notable fue el estímulo que condujo a gran parte de la investigación en los años 90 sobre la búsqueda local y la diferencia entre problemas fáciles y difíciles, los cuales veremos en el Capítulo 7. Hablando de forma aproximada, las *n*-reinas son fáciles para la búsqueda local porque las soluciones están densamente distribuidas en todas las partes del espacio de estados. Los mínimos-conflictos también trabajan bien para problemas difíciles. Por ejemplo, se han utilizado para programar las observaciones del Telescopio Hubble, reduciendo el tiempo utilizado para programar una semana de observaciones, de tres semanas (!) a alrededor de 10 minutos.

Otra ventaja de la búsqueda local consiste en que puede usarse en un ajuste *online* cuando el problema cambia. Esto es particularmente importante en la programación de problemas. El programa de vuelos de una semana puede implicar miles de vuelos y decenas de miles de asignaciones de personal, pero el mal tiempo en un aeropuerto puede convertir a la programación en no factible. Nos gustaría reparar la programación con un número mínimo de cambios. Esto puede hacerse fácilmente con un algoritmo de búsqueda local comenzando desde el programa actual. Una búsqueda con vuelta atrás con un nuevo conjunto de restricciones, por lo general, requiere mucho más tiempo y podría encontrar una solución, desde el programa actual, con muchos cambios.

5.4 La estructura de los problemas

En esta sección, examinamos las formas por las cuales la estructura del problema, representada por el grafo de restricciones, puede utilizarse para encontrar soluciones de forma rápida. La mayor parte de estas aproximaciones son muy generales y aplicables a otros problemas, por ejemplo, razonamiento probabilístico. Después de todo, la única forma que podemos esperar, posiblemente, que pueda tratar con el mundo real es la descomposición en muchos subproblemas. Viendo de nuevo la Figura 5.1(b), con miras a identificar la estructura del problema, se destaca un hecho: Tasmania no está relacionada con el continente[3]. Intuitivamente, es obvio que colorear Tasmania y colorear el continente son **subproblemas independientes** (cualquier solución para el continente combinado con cualquier solución para Tasmania produce una solución para el mapa entero). La independencia puede averiguarse simplemente buscando **componentes conectados** del grafo de restricciones. Cada componente corresponde a un subproblema PSR_i. Si la asignación S_i es una solución de PSR_i, entonces $\cup_i S_i$ es una solución de $\cup_i PSR_i$. ¿Por qué es tan importante? Consideremos lo siguiente: suponga que cada PSR_i tiene c variables del total de n variables, donde c es una constante. Entonces hay n/c subproblemas, cada uno de los cuales trae consigo como mucho d^c de trabajo para resolverlo. De ahí, que el trabajo total es $O(d^c n/c)$, que es lineal en n; sin la descomposición, el trabajo total es $O(d^n)$, que es exponencial en n. Seamos más concretos: la división de un PSR booleano con $n = 80$ en cuatro subproblemas con $c = 20$ reduce el tiempo de resolución, en el caso peor, de la vida del universo hasta menos de un segundo.

SUBPROBLEMAS INDEPENDIENTES

COMPONENTES CONECTADOS

Los subproblemas completamente independientes son deliciosos, pero raros. En la mayoría de casos, los subproblemas de un PSR están relacionados. El caso más simple es cuando el grafo de restricciones forma un **árbol**: cualquiera dos variables están relacionadas por, a lo sumo, un camino. La Figura 5.10(a) muestra un ejemplo esquemático[4].

[3] Un cartógrafo cuidadoso o un tasmanio patriótico podría objetar que Tasmania no debiera ser coloreada lo mismo que su vecino de continente más cercano, lo que evitaría la impresión de que pueda ser parte de aquel estado.

[4] Tristemente, muy pocas regiones del mundo, con la excepción posible de Sulawesi, tienen mapas estructurados por árboles.

Figura 5.10 (a) Grafo de restricciones de un PSR estructurado por árbol. (b) Un ordenamiento lineal de las variables consistentes con el árbol con *A* como la raíz.

Mostraremos que *cualquier PSR estructurado por árbol puede resolverse en tiempo lineal en el número de variables*. El algoritmo tiene los siguientes pasos:

1. Elija cualquier variable como la raíz del árbol, y ordene las variables desde la raíz a las hojas de tal modo que el padre de cada nodo en el árbol lo precede en el ordenamiento. (*Véase* la Figura 5.10(b).) Etiquetar las variables $X_1 \ldots, X_n$ en orden. Ahora, cada variable excepto la raíz tiene exactamente una variable padre.
2. Para j desde n hasta 2, aplicar la consistencia de arco al arco (X_i, X_j), donde X_i es el padre de X_j, quitando los valores del DOMINIO$[X_i]$ que sea necesario.
3. Para j desde 1 a n, asigne cualquier valor para X_j consistente con el valor asignado para X_i, donde X_i es el padre de X_j.

Hay dos puntos claves a destacar. Primero, después del paso 2 el PSR es directamente arco-consistente, entonces la asignación de valores en el paso 3 no requiere ninguna vuelta atrás. (*Véase* la discusión de *k*-consistencia de la página 167.) Segundo, aplicando la comprobación de consistencia de arco en orden inverso en el paso 2, el algoritmo asegura que cualquier valor suprimido no puede poner en peligro la consistencia de arcos que ya han sido tratados. El algoritmo completo se ejecuta en tiempo $O(nd^2)$.

Ahora que tenemos un algoritmo eficiente para árboles, podemos considerar si los grafos restricción más generales pueden *reducirse* a árboles de alguna manera. Hay dos modos primarios de hacer esto, uno basado en quitar nodos y uno basado en nodos que sufren colisiones.

La primera aproximación implica valores de asignación a algunas variables de modo que las variables restantes formen un árbol. Consideremos el grafo restricción para Australia, mostrado otra vez en la Figura 5.11(a). Si pudiéramos suprimir Australia del Sur, el grafo se haría un árbol, como en (b). Por suerte, podemos hacer esto (en el grafo, no en el continente) fijando un valor para *AS* y suprimiendo de los dominios de las otras variables cualquier valor que sea inconsistente con el valor elegido para *AS*.

Ahora, cualquier solución para el PSR después de que *AS* y sus restricciones se quiten, será consistente con el valor elegido para *AS*. (Para los PSRs binarios, la situación es más complicada con restricciones de orden alto.) Por lo tanto, podemos resolver el árbol restante con el algoritmo anterior y así resolver el problema entero. Desde luego, en el caso general (a diferencia de colorear el mapa) el valor elegido para *AS* podría ser

Figura 5.11 (a) Grafo restricción original de la Figura 5.1. (b) Grafo restricción después de quitar *AS*.

el incorrecto, entonces tendríamos que intentarlo con cada uno de ellos. El algoritmo general es como sigue:

CICLO DE CORTE

1. Elegir un subconjunto S de VARIABLES[*psr*] tal que el grafo de restricciones se convierta en un árbol después del quitar S. Llamamos a S un **ciclo de corte**.
2. Para cada asignación posible a las variables en S que satisface todas las restricciones sobre S,
   (a) quitar de los dominios de las variables restantes cualquier valor que sea inconsistente con la asignación para S, y
   (b) si el PSR restante tiene una solución, devolverla junto con la asignación para S.

Si el ciclo de corte tiene tamaño c, entonces el tiempo de ejecución total es $O(d^c \cdot (n - c)d^2)$. Si el grafo es «casi un árbol» entonces c será pequeño y los ahorros sobre la vuelta atrás serán enormes. En el caso peor, sin embargo, c puede ser tan grande como $(n - 2)$. Encontrar el ciclo de corte *más pequeño* es NP-duro, pero se conocen varios algoritmos aproximados eficientes para esta tarea. A la aproximación algorítmica general se le llama **acondicionamiento del corte** que veremos de nuevo en el Capítulo 14, donde se usará para el razonamiento sobre probabilidades.

ACONDICIONAMIENTO DEL CORTE

DESCOMPOSICIÓN EN ÁRBOL

La segunda aproximación está basada en la construcción de una **descomposición en árbol** del grafo restricción en un conjunto de subproblemas relacionados. Cada subproblema se resuelve independientemente, y las soluciones que resultan son entonces combinadas. Como la mayoría de los algoritmos divide-y-vencerás, trabajan bien si ninguno de los subproblemas es demasiado grande. La Figura 5.12 muestra una descomposición de árbol del problema que colorea un mapa en cinco subproblemas. Una descomposición de árbol debe satisfacer las tres exigencias siguientes:

- Cada variable en el problema original aparece en al menos uno de los subproblemas.

- Si dos variables están relacionadas por una restricción en el problema original, deben aparecer juntas (junto con la restricción) en al menos uno de los subproblemas.
- Si una variable aparece en dos subproblemas en el árbol, debe aparecer en cada subproblema a lo largo del camino que une a esos subproblemas.

Las dos primeras condiciones aseguran que todas las variables y las restricciones están representadas en la descomposición. La tercera condición parece bastante técnica, pero simplemente refleja la restricción de que cualquier variable debe tener el mismo valor en cada subproblema en el cual aparece; los enlaces que unen subproblemas en el árbol hacen cumplir esta restricción. Por ejemplo, *AS* aparece en los cuatro subproblemas de la Figura 5.12. Puede verificar de la Figura 5.11 que esta descomposición tiene sentido.

Resolvemos cada subproblema independientemente; si alguno de ellos no tiene ninguna solución, sabemos que el problema entero no tiene ninguna solución. Si podemos resolver todos los subproblemas, entonces intentamos construir una solución global como sigue. Primero, vemos a cada subproblema como «una megavariable» cuyo dominio es el conjunto de todas las soluciones para el subproblema. Por ejemplo, los subproblemas más a la izquierda en la Figura 5.12 forman un problema que colorea el mapa con tres variables y de ahí que tienen seis soluciones (una es $\{AO = rojo, AS = azul, TN = verde\}$). Entonces, resolvemos las restricciones que unen los subproblemas utilizando el algoritmo eficiente para árboles. Las restricciones entre subproblemas simplemente insisten en que las soluciones del subproblema deben estar de acuerdo con sus variables compartidas. Por ejemplo, considerando la solución $\{AO = rojo, AS = azul, TN = verde\}$ para el primer subproblema, la única solución consistente para el siguiente subproblema es $\{AS = azul, TN = verde, Q = rojo\}$.

Figura 5.12 Una descomposición en árbol del grafo restricción de la Figura 5.11(a)

ANCHURA DEL ÁRBOL

El grafo de restricciones admite muchas descomposiciones en árbol; el objetivo en la elección de una descomposición, es hacer los subproblemas tan pequeños como sea posible. La **anchura del árbol** de una descomposición en árbol de un grafo es menor que el tamaño del subproblema más grande; la anchura de árbol del grafo en sí mismo está definida por la anchura de árbol mínimo entre todas sus descomposiciones en árbol. Si un grafo tiene la anchura de árbol w, y nos dan la descomposición en árbol correspondiente, entonces el problema puede resolverse en $O(nd^{w+1})$ veces. De ahí que, los PSRs *con grafos de restricciones de anchura de árbol acotada son resolubles en tiempo polinomial*. Lamentablemente, encontrar la descomposición con la anchura del árbol mínima es NP-duro, pero hay métodos heurísticos que trabajan bien en la práctica.

5.5 Resumen

- Los **problemas de satisfacción de restricciones** (o PSRs) consisten en variables con restricciones sobre ellas. Muchos problemas importantes del mundo real pueden describirse como PSRs. La estructura de un PSR puede representarse por su grafo de restricciones.
- La **búsqueda con vuelta atrás**, una forma de búsqueda primero en profundidad, es comúnmente utilizada para resolver PSRs.
- Las heurísticas de **mínimos valores restantes** y **mínimo grado restante** son métodos, independientes del dominio, para decidir qué variable elegir en una búsqueda con vuelta atrás. La heurística **valor menos restringido** ayuda en la ordenación de los valores de las variables.
- Propagando las consecuencias de las asignaciones parciales que se construyen, el algoritmo con vuelta atrás puede reducir enormemente el factor de ramificación del problema. La **comprobación hacia delante** es el método más simple para hacerlo. La imposición de la **consistencia del arco** es una técnica más poderosa, pero puede ser más costosa de ejecutar.
- La vuelta atrás ocurre cuando no se puede encontrar ninguna asignación legal para una variable. El **salto atrás dirigido por conflictos** vuelve atrás directamente a la fuente del problema.
- La búsqueda local usando la heurística de **mínimos conflictos** se ha aplicado a los problemas de satisfacción de restricciones con mucho éxito.
- La complejidad de resolver un PSR está fuertemente relacionada con la estructura de su grafo de restricciones. Los problemas estructurados por árbol pueden resolverse en tiempo lineal. El **acondicionamiento del corte** puede reducir un PSR general a uno estructurado por árbol y es muy eficiente si puede encontrarse un corte pequeño. Las técnicas de **descomposición en árbol** transforman el PSR en un árbol de subproblemas y son eficientes si la **anchura de árbol** del grafo de restricciones es pequeña.

Notas bibliográficas e históricas

ECUACIONES DIOPHANTINE

El primer trabajo relacionado con la satisfacción de restricciones trató, en gran parte, restricciones numéricas. Las restricciones ecuacionales con dominios de enteros fueron estudiadas por el matemático indio Brahmagupta en el siglo VII; a menudo se les llaman **ecuaciones Diophantine**, después del matemático griego Diophantus (200-284), quien realmente consideró el dominio de racionales positivos. Los métodos sistemáticos para resolver ecuaciones lineales por eliminación de variables fueron estudiados por Gauss (1829); la solución de restricciones lineales en desigualdad se retoman con Fourier (1827).

COLOREO DE UN GRAFO

Los problemas de satisfacción de restricciones con dominios finitos también tienen una larga historia. Por ejemplo, el **coloreo de un grafo** (el coloreo de un mapa es un caso especial) es un viejo problema en matemáticas. Según Biggs *et al*. (1986), la conjetura de cuatro colores (que cada grafo plano puede colorearse con cuatro o menos colores) la hizo primero, en 1852, Fancis Guthrie, un estudiante de Morgan. Esta solución resistió, a pesar de varias reclamaciones publicadas en contra, hasta que Appel y Haken (1977) realizaron una demostración, con la ayuda de un computador.

Clases específicas de problemas de satisfacción de restricciones aparecen a través de la historia de la informática. Uno de los primeros ejemplos más influyentes fue el sistema de SKETCHPAD (Sutherland, 1963), quien resolvió restricciones geométricas en diagramas y fue el precursor de los programas modernos de dibujo y las herramientas CAD. La identificación de PSRs como una clase *general* se debe a Ugo Montanari (1974). La reducción de PSRs de orden alto a PSRs puramente binarios con variables auxiliares (*véase* el Ejercicio 5.11) se debe originalmente al lógico Charles Sanders Peirce del siglo XIX. Fue introducido en la literatura de PSR por Dechter (1990b) y fue elaborado por Bacchus y van Beek (1998). Los PSRs con preferencias entre las soluciones son estudiados ampliamente en la literatura de optimización; *véase* Bistarelli *et al*. (1997) para una generalización del marco de trabajo de PSRs que tienen en cuenta las preferencias. El algoritmo de eliminación de cubo (Dechter, 1999) puede también aplicarse a problemas de optimización.

La búsqueda con vuelta atrás para la satisfacción de restricciones se debe a Bitner y Reingold (1975), aunque ellos remonten el algoritmo básico al siglo XIX. Bitner y Reingold también introdujeron la heurística MVR, que llamaron heurística de la *variable más restringida*. Brelaz (1979) usó el grado heurístico como un modo de deshacer el empate después de aplicar la heurística MVR. El algoritmo que resulta, a pesar de su simplicidad, es todavía el mejor método para el *k*-coloreo de grafos arbitrarios. Haralick y Elliot (1980) propusieron la *heurística del valor menos restringido*.

Los métodos de propagación de restricciones se popularizaron con el éxito de Waltz (1975) sobre problemas poliédricos de etiquetado para la visión por computador. Waltz mostró que, en muchos problemas, la propagación completa elimina la necesidad del retroceso. Montanari (1974) introdujo la noción de redes de restricciones y la propagación por la consistencia del camino. Alan Mackworth (1977) propuso el algoritmo AC-3 para hacer cumplir la consistencia del arco así como la idea general de combinar la vuelta atrás con algún grado de imposición de la consistencia. AC-4, un algoritmo de consistencia

del arco más eficiente, fue desarrollado por Mohr y Henderson (1986). Inmediatamente después de que apareciera el trabajo de Mackworth, los investigadores comenzaron a experimentar sobre la compensación entre el coste al imponer la consistencia y las ventajas en términos de reducción de la búsqueda. Haralick y Elliot (1980) favorecieron el algoritmo de comprobación hacia delante mínima descrito por McGregor (1979), mientras que Gaschnig (1979) sugirió la comprobación completa de la consistencia del arco después de cada asignación de una variable (un algoritmo posterior llamado MAC de Sabin y Freuder (1994)). El trabajo último proporciona pruebas convincentes de que, sobre PSRs más difíciles, la comprobación completa de la consistencia de arco merece la pena. Freuder (1978, 1982) investigó la noción de *k*-consistencia y su relación con la complejidad de resolver PSRs. Apt (1999) describe un marco de trabajo algorítmico genérico dentro del cual se pueden analizar los algoritmos de propagación de consistencia.

Los métodos especiales para manejar restricciones de orden alto se han desarrollado principalmente dentro del contexto de la **programación lógica de restricciones**. Marriot y Stuckey (1998) proporcionan una excelente cobertura de la investigación en este área. La restricción *Todasdif* fue estudiada por Regin (1994). Las restricciones acotadas fueron incorporadas en la programación lógica de restricciones por Van Hentenryck *et al*. (1998).

Los métodos básicos de salto atrás se deben a John Gaschnig (1977, 1979). Kondrak y van Beek (1997) mostraron que este algoritmo es esencialmente subsumido por la comprobación hacia delante. El salto atrás dirigido por conflictos fue ideado por Prosser (1993). La forma más general y poderosa de la vuelta hacia atrás inteligente fue realmente desarrollada muy pronto por Stallman y Sussman (1997). Su técnica del **retroceso dirigido por dependencias** condujo al desarrollo de **sistemas de mantenimiento de la verdad** (Doyle, 1979), de los que hablaremos en la Sección 10.8. La conexión entre las dos áreas la analiza Kleer (1989).

RETROCESO DIRIGIDO POR DEPENDENCIAS

GRABACIÓN DE RESTRICCIONES

El trabajo de Stallman y Sussman también introdujo la idea de la **grabación de restricciones**, en la cual los resultados parciales obtenidos por la búsqueda pueden salvarse y ser reutilizados más tarde. La idea fue introducida formalmente en la búsqueda con vuelta atrás de Dechter (1990a). **Marcar hacia atrás** (Gaschnig, 1979) es un método particularmente simple en el cual se salvan las asignaciones por parejas consistentes e inconsistentes y usadas para evitar comprobar de nuevo las restricciones. Marcar hacia atrás puede combinarse con salto atrás dirigido por conflictos; Kondrad y van Beek (1997) presentan un algoritmo híbrido que probablemente subsume cualquier método. El método del **retroceso dinámico** (Ginsberg, 1993) retiene asignaciones parciales satisfactorias de subconjuntos posteriores de variables cuando volvemos atrás sobre una opción anterior que no invalida el éxito posterior.

MARCAR HACIA ATRÁS

RETROCESO DINÁMICO

La búsqueda local en problemas de satisfacción de restricciones se popularizó con el trabajo de Kirkpatrick *et al*. (1983) sobre el **temple simulado** (*véase* el Capítulo 4), ampliamente utilizado para problemas de programación. El primero que propuso la heurística de mínimo conflicto fue Gu (1989) y desarrollada independientemente por Minton *et al*. (1992). Sosic y Gu (1994) mostraron cómo podría aplicarse para resolver el problema de 3.000.000 de reinas en menos de un minuto. El éxito asombroso de la búsqueda local, usando mínimos conflictos, sobre el problema de las *n*-reinas condujo a una nueva estimación de la naturaleza y predominio de problemas «fáciles» y «difíciles». Peter Cheeseman *et al*. (1991) exploraron la dificultad de los PSRs generados aleatoriamente

y descubrieron que casi todos estos problemas son trivialmente fáciles o no tienen ninguna solución. Sólo si los parámetros del generador del problema se ponen en un cierto rango limitado, dentro del cual aproximadamente la mitad de los problemas son resolubles, encontramos casos de problemas «difíciles». Hablamos de este fenómeno en el Capítulo 7.

El trabajo que relaciona la estructura y la complejidad de los PSRs proviene de Freuder (1985), quien mostró que la búsqueda sobre árboles arco-consistentes trabaja sin ninguna vuelta atrás. Un resultado similar, con extensiones a hipergrafos acíclicos, fue desarrollado en la comunidad de base de datos (Beeri *et al*. 1983). Desde que esos trabajos fueron publicados, hubo mucho progreso en el desarrollo de más resultados generales que relacionan la complejidad de resolver un PSR con la estructura de su grafo de restricciones. La noción de anchura del árbol fue introducida por los teóricos de grafos Robertson y Seymour (1986). Dechter y Pearl (1987, 1989), construyendo sobre el trabajo de Freuder, aplicaron la misma noción (que ellos llamaron la **anchura inducida**) a problemas de satisfacción de restricciones y desarrollaron la aproximación de descomposición en árbol esbozado en la Sección 5.4. Usando este trabajo y sobre resultados de teoría de base de datos, Gottlob *et al*. (1999a, 1999b) desarrollaron una noción, **anchura del hiperárbol**, basada en la caracterización del PSR como un hipergrafo. Además mostraron que cualquier PSR con la anchura de hiperárbol w puede resolverse en tiempo $O(n^{w+1} \log n)$ y que la anchura de hiperárbol subsume todas las medidas de «anchura» (antes definidas) en el sentido de que hay casos donde la anchura de hiperárbol está acotada y las otras medidas no están acotadas.

Hay varias buenas revisiones de técnicas de PSRs, incluyendo la de Kumar (1992), Dechter y Frost (1999), y Bartak (2001); y la enciclopedia de artículos de Dechter (1992) y Mackworth (1992). Pearson y Jeavons (1997) contemplan clases manejables de PSRs, cubriendo tanto métodos de descomposición estructurales como métodos que confían en las propiedades de los dominios o restricciones. Kondrak y van Beek (1997) dan una revisión analítica de algoritmos de búsqueda con vuelta atrás, y Bacchus y van Run (1995) dan una revisión más empírica. Los textos de Tsang (1993) y de Marriott y Stuckey (1998) entran en más profundidad que la realizada en este capítulo. Varias aplicaciones interesantes se describen en la colección editada por Freuder y Mackworth (1994). Los trabajos sobre satisfacción de restricciones aparecen con regularidad en la revista *Artificial Intelligence* y en la revista especializada *Constraints*. La primera conferencia local es la *International Conference on Principles and Practice of Constraint Programming*, amenudo llamada CP.

EJERCICIOS

5.1 Defina con sus propias palabras los términos problema de satisfacción de restricciones, restricción, búsqueda con vuelta atrás, consistencia de arco, salto atrás y mínimos conflictos.

5.2 ¿Cuántas soluciones hay para el problema que colorea el mapa de la Figura 5.1?

5.3 Explique por qué es una buena heurística el elegir la variable que está *más* restringida, en lugar del valor que está menos restringido en una búsqueda de PSR.

5.4 Considere el problema de construir (no resolver) crucigramas[5]: prueba de palabras en una rejilla rectangular. La rejilla, dada como parte del problema, especifica qué cuadrados son en blanco y cuáles sombreados. Asuma que se proporciona una lista de palabras (es decir, un diccionario) y que la tarea es llenar los cuadrados en blanco usando cualquier subconjunto de la lista. Formule este problema, con precisión, de dos modos:

a) Como un problema general de búsqueda. Elija un algoritmo de búsqueda apropiado, y especifique una función heurística, si piensa que es necesaria. ¿Es mejor rellenar con una letra o una palabra a la vez?

b) Como un problema de satisfacción de restricciones ¿deberían las variables ser palabras o letras?

¿Qué formulación piensa que será mejor? ¿Por qué?

5.5 Dé formulaciones precisas para cada uno de los siguientes problemas de satisfacción de restricciones:

PLANIFICACIÓN EN EL PLANO

PLANIFICACIÓN DE CLASES

a) **Planificación en el plano** rectilíneo: encuentre lugares no solapados en un rectángulo grande para varios rectángulos más pequeños.

b) **Programación de clases**: Hay un número fijo de profesores y aulas, una lista de clases, y una lista de huecos posibles para las clases. Cada profesor tiene un conjunto de clases que él o ella pueden enseñar.

5.6 Resuelva a mano el problema criptoaritmético de la Figura 5.2, usando el retroceso, comprobación hacia delante, y las heurísticas MVR y la del valor menos restringido.

5.7 La Figura 5.5 prueba varios algoritmos sobre el problema de las n-reinas. Intente estos mismos algoritmos sobre problemas de coloreo de mapas generados aleatoriamente como sigue: disperse n puntos sobre el cuadrado unidad; seleccionando un punto X al azar, una X con una línea recta al punto más cercano Y tal que X no está ya relacionado con Y y la línea no cruce ninguna otra línea; repita el paso anterior hasta que no haya más uniones posibles. Construya la tabla de funcionamiento para el n más grande que pueda manejar, usando tanto colores $d = 4$ como $d = 3$. Comente sus resultados.

5.8 Utilice el algoritmo AC-3 para mostrar que la consistencia de arco es capaz de descubrir la inconsistencia de la asignación parcial $\{AO = rojo, V = azul\}$ para el problema de la Figura 5.1.

5.9 ¿Cuál es la complejidad, en el caso peor, al ejecutar AC-3 sobre un PSR estructurado por árbol?

5.10 AC-3 vuelve a poner sobre la cola cada arco (X_k, X_i) siempre que *algún* valor sea suprimido del dominio de X_i, aunque cada valor de X_k sea consistente con los valores restantes de X_i. Suponga que, para cada arco (X_k, X_i), guardamos el número de valores restantes de X_i que son consistentes con cada valor de X_k. Explique cómo actualizar estos números de manera eficiente y muestre que la consistencia de arco puede hacerse en tiempo total $O(n^2d^2)$.

---

[5] Ginsberg *et al*. (1990) hablan de varios métodos para construir crucigramas. Littman *et al*. (1999) abordan el problema más difícil: resolverlos.

5.11 Muestre cómo una restricción ternaria simple como «$A + B = C$» puede transformarse en tres restricciones binarias usando una variable auxiliar. Puede suponer dominios finitos. (*Consejo*: considere una nueva variable que tome valores que son pares de otros valores, y considere que las restricciones como «X es el primer elemento del par Y».) Después, muestre cómo las restricciones con más de tres variables pueden tratarse de modo similar. Finalmente, muestre cómo las restricciones con una sola variable pueden eliminarse cambiando los dominios de las variables. Esto completa la demostración de que cualquier PSR puede transformarse en un PSR con restricciones binarias.

5.12 Suponga que un grafo tiene un ciclo de corte de no más de k nodos. Describa un algoritmo simple para encontrar un ciclo de corte mínimo cuyo tiempo de ejecución no sea mucho más que $O(n^k)$ para un PSR con n variables. Busque en la literatura métodos para encontrar, aproximadamente, el ciclo de corte mínimo en tiempo polinomial en el tamaño del corte. ¿La existencia de tales algoritmos hacen práctico al método de ciclo de corte?

5.13 Considere el siguiente puzle lógico: en cinco casas, cada una con un color diferente, viven cinco personas de nacionalidades diferentes, cada una de las cuales prefiere una marca diferente de cigarrillos, una bebida diferente, y un animal doméstico diferente. Considerando los hechos siguientes, la pregunta para contestar es «¿dónde vive Zebra, y en qué casa beben ellos el agua?»

El inglés vive en la casa roja.
El español posee el perro.
El noruego vive en la primera casa a la izquierda.
Los Kools son fumados en la casa amarilla.
El hombre que fuma Chesterfields vive en la casa al lado del hombre con el zorro.
El noruego vive al lado de la casa azul.
El fumador de Winston posee caracoles.
El fumador de Lucky Strike bebe zumo de naranja.
El ucraniano bebe el té.
El japonés fuma los Parliaments.
Los Kools son fumados en la casa al lado de la casa donde se guarda el caballo.
El café es bebido en la casa verde.
La casa verde está inmediatamente a la derecha (su derecha) de la casa de color marfil.
La leche es bebida en la casa del medio.

Discuta diferentes representaciones de este problema como un PSR. ¿Por qué preferiría una representación sobre otra?

6 Búsqueda entre adversarios

Donde examinaremos los problemas que surgen cuando tratamos de planear en un mundo donde otros agentes planean contra nosotros.

6.1 Juegos

En el Capítulo 2 se introdujeron los **entornos multiagente**, en los cuales cualquier agente tendrá que considerar las acciones de otros agentes y cómo afectan a su propio bienestar. La imprevisibilidad de estos otros agentes puede introducir muchas posibles **contingencias** en el proceso de resolución de problemas del agente, como se discutió en el Capítulo 3. En el Capítulo 2 también se introdujo la diferencia entre entornos multiagente **cooperativos** y **competitivos**. Los entornos competitivos, en los cuales los objetivos del agente están en conflicto, dan ocasión a problemas de **búsqueda entre adversarios**, a menudo conocidos como **juegos**.

JUEGOS

La **teoría matemática de juegos**, una rama de la Economía, ve a cualquier entorno multiagente como un juego a condición de que el impacto de cada agente sobre los demás sea «significativo», sin tener en cuenta si los agentes son cooperativos o competitivos[1]. En IA, «los juegos» son, por lo general, una clase más especializada (que los teóricos de juegos llaman **juegos de suma cero,** de dos jugadores, por turnos, determinista, de **información perfecta**). En nuestra terminología, significan entornos deterministas, totalmente observables en los cuales hay dos agentes cuyas acciones deben alternar y en los que los valores utilidad, al final de juego, son siempre iguales y opuestos. Por ejemplo, si un jugador gana un juego de ajedrez (+1), el otro jugador necesa-

JUEGOS DE SUMA CERO

INFORMACIÓN PERFECTA

[1] Los entornos con muchos agentes se ven mejor como entornos **económicos** más que como juegos.

riamente pierde (−1). Esta oposición entre las funciones de utilidad de los agentes hace la situación entre adversarios. Consideraremos brevemente en este capítulo juegos multi-jugador, juegos de suma no cero, y juegos estocásticos, pero aplazaremos hasta el Capítulo 17 la discusión de la teoría de juegos apropiada.

Los juegos han ocupado las facultades intelectuales de la gente (a veces a un grado alarmante) mientras ha existido la civilización. Para los investigadores de IA, la naturaleza abstracta de los juegos los hacen un tema atractivo a estudiar. El estado de un juego es fácil de representar, y los agentes están restringidos, por lo general, a un pequeño número de acciones cuyos resultados están definidos por reglas precisas. Los juegos físicos, como croquet y hockey sobre hielo, tienen descripciones mucho más complicadas, una variedad mucho más grande de acciones posibles, y reglas bastante imprecisas que definen la legalidad de las acciones. A excepción del fútbol de robots, estos juegos físicos no han tenido mucho interés en la comunidad de IA.

El jugar a juegos fue una de las primeras tareas emprendidas en IA. Hacia 1950, casi tan pronto como los computadores se hicieron programables, el ajedrez fue abordado por Konrad Zuse (el inventor del primer computador programable y del primer lenguaje de programación), por Claude Shannon (el inventor de la teoría de información), por Norbert Wiener (el creador de la teoría de control moderna), y por Alan Turing. Desde entonces, hubo un progreso continuo en el nivel de juego, hasta tal punto que las máquinas han superado a la gente en las damas y en Otelo, han derrotado a campeones humanos (aunque no siempre) en ajedrez y *backgammon*, y son competitivos en muchos otros juegos. La excepción principal es Go, en el que los computadores funcionan a nivel aficionado.

Los juegos, a diferencia de la mayor parte de los problemas de juguete estudiados en el Capítulo 3, son interesantes *porque* son demasiado difíciles para resolverlos. Por ejemplo, el ajedrez tiene un factor de ramificación promedio de aproximadamente 35, y los juegos a menudo van a 50 movimientos por cada jugador, entonces el árbol de búsqueda tiene aproximadamente 35^{100} o 10^{154} nodos (aunque el grafo de búsqueda tenga «sólo» aproximadamente 10^{40} nodos distintos). Por lo tanto, los juegos, como el mundo real, requieren la capacidad de tomar *alguna* decisión cuando es infactible calcular la decisión *óptima*. Los juegos también castigan la ineficiencia con severidad. Mientras que una implementación de la búsqueda A* que sea medio eficiente costará simplemente dos veces más para ejecutarse por completo, un programa de ajedrez que sea medio eficiente en la utilización de su tiempo disponible, probablemente tendrá que descartarse si no intervienen otros factores. La investigación en juegos ha generado, por lo tanto, varias ideas interesantes sobre cómo hacer uso, lo mejor posible, del tiempo.

Comenzamos con una definición del movimiento óptimo y un algoritmo para encontrarlo. Veremos técnicas para elegir un movimiento bueno cuando el tiempo es limitado. La **poda** nos permite ignorar partes del árbol de búsqueda que no marcan ninguna diferencia para obtener la opción final, y las **funciones de evaluación** heurísticas nos permiten aproximar la utilidad verdadera de un estado sin hacer una búsqueda completa. La Sección 6.5 habla de juegos como el backgammon que incluyen un elemento de posibilidad; también hablamos del bridge, que incluye elementos de **información imperfecta** al no ser visibles todas las cartas a cada jugador. Finalmente, veremos cómo se desenvuelven los programas de juegos contra la oposición humana y las direcciones para el desarrollo futuro.

INFORMACIÓN IMPERFECTA

6.2 Decisiones óptimas en juegos

Consideraremos juegos con dos jugadores, que llamaremos MAX y MIN por motivos que pronto se harán evidentes. MAX mueve primero, y luego mueven por turno hasta que el juego se termina. Al final de juego, se conceden puntos al jugador ganador y penalizaciones al perdedor. Un juego puede definirse formalmente como una clase de problemas de búsqueda con los componentes siguientes:

- El **estado inicial**, que incluye la posición del tablero e identifica al jugador que mueve.
- Una **función sucesor**, que devuelve una lista de pares (*movimiento, estado*), indicando un movimiento legal y el estado que resulta.

TEST TERMINAL

- Un **test terminal**, que determina cuándo se termina el juego. A los estados donde el juego se ha terminado se les llaman **estados terminales**.
- Una **función utilidad** (también llamada función objetivo o función de rentabilidad), que da un valor numérico a los estados terminales. En el ajedrez, el resultado es un triunfo, pérdida, o empate, con valores $+1$, -1 o 0. Algunos juegos tienen una variedad más amplia de resultados posibles: las rentabilidades en el *backgammon* se extienden desde $+192$ a -192. Este capítulo trata principalmente juegos de suma cero, aunque mencionemos brevemente juegos con «suma no cero».

ÁRBOL DE JUEGOS

El estado inicial y los movimientos legales para cada lado definen el **árbol de juegos**. La Figura 6.1 muestra la parte del árbol de juegos para el tic-tac-toe (tres en raya). Desde el estado inicial, MAX tiene nueve movimientos posibles. El juego alterna entre la colocación de una X para MAX y la colocación de un O para MIN, hasta que alcancemos nodos hoja correspondientes a estados terminales, de modo que un jugador tenga tres en raya o todos los cuadrados estén llenos. El número sobre cada nodo hoja indica el valor de utilidad del estado terminal desde el punto de vista de MAX; se supone que los valores altos son buenos para MAX y malos para MIN (por eso los nombres de los jugadores). Este trabajo de MAX al usar el árbol de búsqueda (en particular la utilidad de estados terminales) determina el mejor movimiento.

Estrategias óptimas

En un problema de búsqueda normal, la solución óptima sería una secuencia de movimientos que conducen a un estado objetivo (un estado terminal que es ganador). En un juego, por otra parte, MIN tiene algo que decir sobre ello. MAX por lo tanto debe encontrar una **estrategia** contingente, que especifica el movimiento de MAX en el estado inicial, después los movimientos de MAX en los estados que resultan de cada respuesta posible de MIN, después los movimientos de MAX en los estados que resultan de cada respuesta posible de MIN de los *anteriores* movimientos, etcétera. Hablando de forma aproximada, una estrategia óptima conduce a resultados al menos tan buenos como cualquier otra estrategia cuando uno juega con un oponente infalible. Comenzaremos mostrando cómo encontrar esta estrategia óptima, aunque será infactible, para MAX, al calcularla en juegos más complejos que tic-tac-toe.

ESTRATEGIA

Figura 6.1 Un árbol (parcial) de búsqueda para el juego de tic-tac-toe. El nodo de arriba es el estado inicial, y MAX mueve primero, colocando un X en un cuadrado vacío. Mostramos la parte del árbol de búsqueda, con movimientos alternados por MIN (O) y MAX, hasta que finalmente alcancemos estados terminales, en los cuales podemos asignar utilidades según las reglas del juego.

Incluso un juego simple como tic-tac-toe es demasiado complejo para dibujar el árbol de juegos entero, por tanto cambiemos al juego trivial de la Figura 6.2. Los movimientos posibles para MAX, en el nodo raíz, se etiquetan por a_1, a_2, y a_3. Las respuestas posibles a a_1, para MIN, son b_1, b_2, b_3, etc. Este juego particular finaliza después de un movimiento para MAX y MIN. (En el lenguaje de juegos, decimos que este árbol es un movimiento en profundidad, que consiste en dos medios movimientos, cada uno de los cuales se llama **capa**.) Las utilidades de los estados terminales en este juego varía desde dos a 14.

CAPA

Considerando un árbol de juegos, la estrategia óptima puede determinarse examinando el **valor minimax** de cada nodo, que escribimos como el VALOR-MINIMAX(n). El valor minimax de un nodo es la utilidad (para MAX) de estar en el estado correspondiente, *asumiendo que ambos jugadores juegan óptimamente* desde allí al final del juego. Obviamente, el valor minimax de un estado terminal es solamente su utilidad. Además, considerando una opción, MAX preferirá moverse a un estado de valor máximo, mientras que MIN prefiere un estado de valor mínimo. Entonces tenemos lo siguiente:

VALOR MINIMAX

$$\text{VALOR-MINIMAX}(n) = \begin{cases} \text{UTILIDAD}(n) & \text{si } n \text{ es un estado terminal} \\ \max_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MAX} \\ \min_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MIN} \end{cases}$$

Figura 6.2 Un árbol de juegos de dos capas. Los nodos △ son «nodos MAX», en los que le toca mover a MAX, y los nodos ▽ son «nodos MIN». Los nodos terminales muestran los valores de utilidad para MAX; los otros nodos son etiquetados por sus valores minimax. El mejor movimiento de MAX en la raíz es a_1, porque conduce al sucesor con el valor minimax más alto, y la mejor respuesta de MIN es b_1, porque conduce al sucesor con el valor minimax más bajo.

Apliquemos estas definiciones al árbol de juegos de la Figura 6.2. Los nodos terminales se etiquetan por sus valores de utilidad. El primer nodo de MIN, etiquetado *B*, tiene tres sucesores con valores 3, 12 y 8, entonces su valor minimax es 3. Del mismo modo, los otros dos nodos de MIN tienen un valor minimax de 2. El nodo raíz es un nodo MAX; sus sucesores tienen valores minimax de 3, 2 y 2; entonces tiene un valor minimax de 3. Podemos identificar también la **decisión minimax** en la raíz: la acción a_1 es la opción óptima para MAX porque conduce al sucesor con el valor minimax más alto.

DECISIÓN MINIMAX

Esta definición de juego óptimo para MAX supone que MIN también juega óptimamente (maximiza los resultados del *caso-peor* para MAX). ¿Y si MIN no juega óptimamente? Entonces es fácil demostrar (Ejercicio 6.2) que MAX lo hará aún mejor. Puede haber otras estrategias contra oponentes subóptimos que lo hagan mejor que la estrategia minimax; pero estas estrategias necesariamente lo hacen peor contra oponentes óptimos.

El algoritmo minimax

ALGORITMO MINIMAX

El **algoritmo minimax** (Figura 6.3) calcula la decisión minimax del estado actual. Usa un cálculo simple recurrente de los valores minimax de cada estado sucesor, directamente implementando las ecuaciones de la definición. La recursión avanza hacia las hojas del árbol, y entonces los valores minimax **retroceden** por el árbol cuando la recursión se va deshaciendo. Por ejemplo, en la Figura 6.2, el algoritmo primero va hacia abajo a los tres nodos izquierdos, y utiliza la función UTILIDAD para descubrir que sus valores son 3, 12 y 8 respectivamente. Entonces toma el mínimo de estos valores, 3, y lo devuelve como el valor del nodo *B*. Un proceso similar devuelve hacia arriba el valor de 2 para *C* y 2 para *D*. Finalmente, tomamos el máximo de 3, 2 y 2 para conseguir el valor de 3 para el nodo de raíz.

RETROCEDER

El algoritmo minimax realiza una exploración primero en profundidad completa del árbol de juegos. Si la profundidad máxima del árbol es *m*, y hay *b* movimientos legales

```
función DECISIÓN-MINIMAX(estado) devuelve una acción
  variables de entrada: estado, estado actual del juego

  v ← MAX-VALOR(estado)
  devolver la acción de SUCESORES(estado) con valor v
```

```
función MAX-VALOR(estado) devuelve un valor utilidad
  si TEST-TERMINAL(estado) entonces devolver UTILIDAD(estado)
  v ← −∞
  para un s en SUCESORES(estado) hacer
    v ← MAX(v, MIN-VALOR(s))
  devolver v
```

```
función MAX-VALOR(estado) devuelve un valor utilidad
  si TEST-TERMINAL(estado) entonces devolver UTILIDAD(estado)
  v ← ∞
  para un s en SUCESORES(estado) hacer
    v ← MIN(v, MAX-VALOR(s))
  devolver v
```

Figura 6.3 Un algoritmo para el cálculo de decisiones minimax. Devuelve la acción correspondiente al movimiento mejor posible, es decir, el movimiento que conduce al resultado con la mejor utilidad, conforme al axioma que el oponente juega para minimizar la utilidad. Las funciones VALOR-MAX y el VALOR-MIN pasan por el árbol de juegos entero, por todos los caminos hacia las hojas, para determinar el valor que le llega a un estado.

en cada punto, entonces la complejidad en tiempo del algoritmo minimax es $O(b^m)$. La complejidad en espacio es $O(bm)$ para un algoritmo que genere todos los sucesores a la vez, o $O(m)$ para un algoritmo que genere los sucesores uno por uno (*véase* la página 86). Para juegos reales, desde luego, los costos de tiempo son totalmente poco prácticos, pero este algoritmo sirve como base para el análisis matemático de juegos y para algoritmos más prácticos.

Decisiones óptimas en juegos multi-jugador

Muchos juegos populares permiten más de dos jugadores. Examinemos cómo obtener una extensión de la idea minimax a juegos multi-jugador. Esto es sencillo desde el punto de vista técnico, pero proporciona algunas nuevas cuestiones conceptuales interesantes.

Primero, tenemos que sustituir el valor para cada nodo con un *vector* de valores. Por ejemplo, en un juego de tres jugadores con jugadores A, B y C, un vector $\langle v_A, v_B, v_C \rangle$ asociado con cada nodo. Para los estados terminales, este vector dará la utilidad del estado desde el punto de vista de cada jugador. (En dos jugadores, en juegos de suma cero, el vector de dos elementos puede reducirse a un valor porque los valores son siempre opuestos.) El camino más simple de implementarlo es hacer que la función UTILIDAD devuelva un vector de utilidades.

Ahora tenemos que considerar los estados no terminales. Consideremos el nodo marcado con X en el árbol de juegos de la Figura 6.4. En ese estado, el jugador C elige qué hacer. Las dos opciones conducen a estados terminales con el vector de utilidad $\langle v_A = 1, v_B = 2, v_C = 6\rangle$ y $\langle v_A = 4, v_B = 2, v_C = 3\rangle$. Como 6 es más grande que 3, C debería elegirlo como primer movimiento. Esto significa que si se alcanza el estado X, el movimiento siguiente conducirá a un estado terminal con utilidades $\langle v_A = 1, v_B = 2, v_C = 6\rangle$. De ahí, que el valor que le llega a X es este vector. En general, el valor hacia atrás de un nodo n es el vector de utilidad de cualquier sucesor que tiene el valor más alto para el jugador que elige en n.

ALIANZAS

Alguien que juega juegos multi-jugador, como Diplomacy™, rápidamente se da cuenta de que hay muchas más posibilidades que en los juegos de dos jugadores. Los juegos multi-jugador, por lo general, implican **alianzas**, formales o informales, entre los jugadores. Se hacen y se rompen las alianzas conforme avanza el juego. ¿Cómo debemos entender tal comportamiento? ¿Las alianzas son una consecuencia natural de estrategias óptimas para cada jugador en un juego multi-jugador? Resulta que pueden ser. Por ejemplo suponga que A y B están en posiciones débiles y C está en una posición más fuerte. Entonces, a menudo, es óptimo tanto para A como para B atacar a C más que el uno al otro, por temor a que C los destruya. De esta manera, la colaboración surge del comportamiento puramente egoísta. Desde luego, tan pronto como C se debilita bajo el ataque conjunto, la alianza pierde su valor, y A o B podrían violar el acuerdo. En algunos casos, las alianzas explícitas solamente concretan lo que habría pasado de todos modos. En otro caso hay un estigma social a la rotura de una alianza, así que los jugadores deben equilibrar la ventaja inmediata de romper una alianza contra la desventaja a largo plazo de ser percibido como poco fiable. *Véase* la Sección 17.6 para estas complicaciones.

Si el juego no es de suma cero, la colaboración puede ocurrir también con sólo dos jugadores. Suponga, por ejemplo, que hay un estado terminal con utilidades $\langle v_A = 1.000, v_B = 1.000\rangle$, y que 1.000 es la utilidad más alta posible para cada jugador. Entonces la estrategia óptima para ambos jugadores es hacer todo lo posible por alcanzar este estado (es decir los jugadores cooperarán automáticamente para conseguir un objetivo mutuamente deseable).

Figura 6.4 Las tres primeras capas de un árbol de juegos con tres jugadores (A, B, C). Cada nodo se etiqueta con valores desde el punto de vista de cada jugador. En la raíz se marca el mejor movimiento.

6.3 Poda alfa-beta

El problema de la búsqueda minimax es que el número de estados que tiene que examinar es exponencial en el número de movimientos. Lamentablemente no podemos eliminar el exponente, pero podemos dividirlo, con eficacia, en la mitad. La jugada es que es posible calcular la decisión minimax correcta sin mirar todos los nodos en el árbol de juegos. Es decir podemos tomar prestada la idea de **podar** del Capítulo 4 a fin de

Figura 6.5 Etapas en el cálculo de la decisión óptima para el árbol de juegos de la Figura 6.2. En cada punto, mostramos el rango de valores posibles para cada nodo. (a) La primera hoja debajo de *B* tiene el valor 3. De ahí *B*, que es un nodo MIN, tiene un valor de *como máximo* 3. (b) La segunda hoja debajo de *B* tiene un valor de 12; MIN evitaría este movimiento, entonces el valor de *B* es todavía como máximo 3. (c) La tercera hoja debajo de *B* tiene un valor de 8; hemos visto a todos los sucesores de *B*, entonces el valor de *B* es exactamente 3. Ahora, podemos deducir que el valor de la raíz es *al menos* 3, porque MAX tiene una opción digna de 3 en la raíz. (d) La primera hoja debajo de *C* tiene el valor 2. De ahí, *C*, que es un nodo MIN, tiene un valor de *como máximo* 2. Pero sabemos que *B* vale 3, entonces MAX nunca elegiría *C*. Por lo tanto, no hay ninguna razón en mirar a los otros sucesores de *C*. Este es un ejemplo de la poda de alfa-beta. (e) La primera hoja debajo de *D* tiene el valor 14, entonces *D* vale *como máximo* 14. Este es todavía más alto que la mejor alternativa de MAX (es decir, 3), entonces tenemos que seguir explorando a los sucesores de *D*. Note también que ahora tenemos límites sobre todos los sucesores de la raíz, entonces el valor de la raíz es también como máximo 14. (f) El segundo sucesor de *D* vale 5, así que otra vez tenemos que seguir explorando. El tercer sucesor vale 2, así que ahora *D* vale exactamente 2. La decisión de MAX en la raíz es moverse a *B*, dando un valor de 3.

PODA ALFA-BETA

eliminar partes grandes del árbol. A la técnica que examinaremos se le llama **poda alfa-beta**. Cuando lo aplicamos a un árbol minimax estándar, devuelve el mismo movimiento que devolvería minimax, ya que podar las ramas no puede influir, posiblemente, en la decisión final.

Consideremos otra vez el árbol de juegos de dos capas de la Figura 6.2. Vamos a realizar el cálculo de la decisión óptima una vez más, esta vez prestando atención a lo que sabemos en cada punto del proceso. En la Figura 6.5 se explican los pasos. El resultado es que podemos identificar la decisión minimax sin evaluar dos de los nodos hoja.

Otro modo de verlo es como una simplificación de la fórmula VALOR-MINIMAX. Los dos sucesores no evaluados del nodo C de la Figura 6.5 tienen valores x e y, y sea z el mínimo entre x e y. El valor del nodo raíz es

$$\begin{aligned}\text{MINIMAX-VALUE}(\textit{raíz}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \qquad \text{donde } z \leq 2 \\ &= 3\end{aligned}$$

En otras palabras, el valor de la raíz y de ahí la decisión mínimax son *independientes* de los valores de las hojas podadas x e y.

La poda alfa-beta puede aplicarse a árboles de cualquier profundidad, y, a menudo, es posible podar subárboles enteros. El principio general es: considere un nodo n en el árbol (*véase* la Figura 6.6), tal que el Jugador tiene una opción de movimiento a ese nodo. Si el Jugador tiene una mejor selección m en el nodo padre de n o en cualquier punto más lejano, entonces *n nunca será alcanzado en el juego actual*. Una vez que hemos averiguado bastante sobre n (examinando a algunos de sus descendientes) para alcanzar esta conclusión, podemos podarlo.

Recuerde que la búsqueda minimax es primero en profundidad, así que en cualquier momento solamente tenemos que considerar los nodos a lo largo de un camino en el árbol.

Figura 6.6 Poda alfa-beta: el caso general. Si m es mejor que n para el Jugador, nunca iremos a n en el juego.

La poda alfa-beta consigue su nombre de los dos parámetros que describen los límites sobre los valores hacia atrás que aparecen a lo largo del camino:

α = el valor de la mejor opción (es decir, valor más alto) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MAX.

β = el valor de la mejor opción (es decir, valor más bajo) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MIN.

La búsqueda alfa-beta actualiza el valor de α y β según se va recorriendo el árbol y poda las ramas restantes en un nodo (es decir, termina la llamada recurrente) tan pronto como el valor del nodo actual es peor que el actual valor α o β para MAX o MIN, respectivamente. La Figura 6.7 nos da el algoritmo completo. Animamos al lector a trazar su comportamiento cuando lo aplicamos al árbol de la Figura 6.5.

La eficacia de la poda alfa-beta es muy dependiente del orden en el que se examinan los sucesores. Por ejemplo, en la Figura 6.5(e) y (f), podríamos no podar ningún

```
función BÚSQUEDA-ALFA-BETA(estado) devuelve una acción
  variables de entrada: estado, estado actual del juego

  v ← MAX-VALOR(estado, −∞, +∞)
  devolver la acción de SUCESORES(estado) con valor v
```

```
función MAX-VALOR(estado,α, β) devuelve un valor utilidad
  variables de entrada: estado, estado actual del juego
            α, valor de la mejor alternativa para MAX a lo largo del camino a estado
            β, valor de la mejor alternativa para MIN a lo largo del camino a estado

  si TEST-TERMINAL(estado) entonces devolver UTILIDAD(estado)
  v ← −∞
  para a, s en SUCESORES(estado) hacer
  v ← MAX(V, MIN-VALOR(s, α, β))
      si v ≥ β entonces devolver v
      α ← MAX(α, v)
  devolver v
```

```
función MIN-VALOR(estado, α, β) devuelve un valor utilidad
  variables de entrada: estado, estado actual del juego
            α, valor de la mejor alternativa para MAX a lo largo del camino a estado
            β, valor de la mejor alternativa para MIN a lo largo del camino a estado

  si TEST-TERMINAL(estado) entonces devolver UTILIDAD(estado)
  v ← +∞
  para a, s en SUCESORES(estado) hacer
      v ← MIN(v, MAX-VALOR(s, α, β))
      si v ≤ α entonces devolver v
      β ← MIN(β, V)
  devolver v
```

Figura 6.7 El algoritmo de búsqueda alfa-beta. Notemos que estas rutinas son lo mismo que las rutinas de MINIMAX de la Figura 6.3, excepto las dos líneas MIN-VALOR y MAX-VALOR que mantienen α y β (y la forma de hacer pasar estos parámetros).

sucesor de D si se hubieran generado primero los sucesores peores (desde el punto de vista de MIN). Si el tercer sucesor se hubiera generado primero, habríamos sido capaces de podar los otros dos. Esto sugiere que pudiera valer la pena tratar de examinar primero los sucesores que probablemente serán los mejores.

Si asumimos que esto puede hacerse[2], resulta que alfa-beta tiene que examinar sólo $O(b^{d/2})$ nodos para escoger el mejor movimiento, en vez de $O(b^d)$ para minimax. Esto significa que el factor de ramificación eficaz se hace $\sqrt{b}$ en vez de b (para el ajedrez, seis en vez de 35). De otra manera, alfa-beta puede mirar hacia delante aproximadamente dos veces más que minimax en la misma cantidad del tiempo. Si los sucesores se examinan en orden aleatorio más que primero el mejor, el número total de nodos examinados será aproximadamente $O(b^{3d/4})$ para un moderado b. Para el ajedrez, una función de ordenación bastante sencilla (como intentar primero las capturas, luego las amenazas, luego mover hacia delante, y, por último, los movimientos hacia atrás) lo consiguen en aproximadamente un factor de dos del resultado $O(b^{d/2})$ del mejor caso. Añadir esquemas dinámicos que ordenan movimientos, como intentar primero los movimientos que fueron mejores la última vez, llegamos bastante cerca del límite teórico.

En el Capítulo 3, vimos que los estados repetidos en el árbol de búsqueda pueden causar un aumento exponencial del coste de búsqueda. En juegos, los estados repetidos ocurren con frecuencia debido a **transposiciones** (permutaciones diferentes de la secuencia de movimientos que terminan en la misma posición). Por ejemplo, si Blanco tiene un movimiento a_1 que puede ser contestado por Negro con b_1 y un movimiento no relacionado a_2 del otro lado del tablero puede ser contestado por b_2, entonces las secuencias $[a_1, b_1, a_2, b_2]$ y $[a_1, b_2, a_2, b_1]$ terminan en la misma posición (como las permutaciones que comienzan con a_2). Vale la pena almacenar la evaluación de esta posición en una tabla *hash* la primera vez que se encuentre, de modo que no tuviéramos que volver a calcularlo las siguientes veces. Tradicionalmente a la tabla *hash* de posiciones se le llama **tabla de transposición**; es esencialmente idéntica a la lista *cerrada* en la BÚSQUEDA-GRAFOS (página 93). La utilización de una tabla de transposiciones puede tener un efecto espectacular a veces, tanto como doblar la profundidad accesible de búsqueda en el ajedrez. Por otra parte, si evaluamos un millón de nodos por segundo, no es práctico guardar *todos* ellos en la tabla de transposiciones. Se han utilizado varias estrategias para elegir los más valiosos.

TRANSPOSICIONES

TABLA DE TRANSPOSICIONES

6.4 Decisiones en tiempo real imperfectas

El algoritmo minimax genera el espacio de búsqueda entero, mientras que el algoritmo alfa-beta permite que podemos partes grandes de él. Sin embargo, alfa-beta todavía tiene que buscar en todos los caminos, hasta los estados terminales, para una parte del espacio de búsqueda. Esta profundidad no es, por lo general, práctica porque los movimientos deben hacerse en una cantidad razonable de tiempo (típicamente, unos minutos como máximo). El trabajo de Shannon en 1950, *Programming a computer for playing chess*, pro-

[2] Obviamente, no puede hacerse; por otra parte ¡la función de ordenación podría usarse para jugar un juego perfecto!

puso un cambio: que los programas deberían cortar la búsqueda antes y entonces aplicar una **función de evaluación** heurística a los estados, convirtiendo, efectivamente, los nodos no terminales en hojas terminales. En otras palabras, la sugerencia deberá alterar minimax o alfa-beta de dos formas: se sustituye la función de utilidad por una función de evaluación heurística EVAL, que da una estimación de la utilidad de la posición, y se sustituye el test-terminal por un **test-límite** que decide cuándo aplicar EVAL.

TEST-LÍMITE

Funciones de evaluación

Una función de evaluación devuelve una estimación de la utilidad esperada de una posición dada, tal como hacen las funciones heurísticas del Capítulo 4 que devuelven una estimación de la distancia al objetivo. La idea de una estimación no era nueva cuando Shannon la propuso. Durante siglos, los jugadores de ajedrez (y aficionados de otros juegos) han desarrollado modos de juzgar el valor de una posición, debido a que la gente es aún más limitada, en cantidad de la búsqueda, que los programas de computador. Debería estar claro que el funcionamiento de un programa de juegos es dependiente de la calidad de su función de evaluación. Una función de evaluación inexacta dirigirá a un agente hacia posiciones que resultan estar perdidas. ¿Cómo diseñamos funciones de evaluación buenas?

Primero, la función de evaluación debería ordenar los estados *terminales* del mismo modo que la función de utilidad verdadera; por otra parte, un agente que la use podría seleccionar movimientos subóptimos aunque pueda ver delante el final del juego. Segundo, ¡el cálculo no debe utilizar demasiado tiempo! (La función de evaluación podría llamar a la DECISIÓN-MINIMAX como una subrutina y calcular el valor exacto de la posición, pero esto frustraría nuestro propósito: ahorrar tiempo.) Tercero, para estados no terminales, la función de evaluación debería estar fuertemente correlacionada con las posibilidades actuales de ganar.

Uno podría preguntarse sobre la frase «las posibilidades de ganar». Después de todo, el ajedrez no es un juego de azar: sabemos el estado actual con certeza. Pero si la búsqueda debe cortarse en estados no terminales, entonces necesariamente el algoritmo será *incierto* sobre los resultados finales de esos estados. Este tipo de incertidumbre está inducida por limitaciones computacionales, más que de información. Si consideramos la cantidad limitada de cálculo que se permiten a la función de evaluación cuando se aplica a un estado, lo mejor que se podría hacer es una conjetura sobre el resultado final.

CARACTERÍSTICAS

Hagamos esta idea más concreta. La mayoría de las funciones de evaluación trabajan calculando varias **características** del estado (por ejemplo, en el juego de ajedrez, el número de peones capturados por cada lado). Las características, juntas, definen varias *categorías* o *clases de equivalencia* de estados: los estados en cada categoría tienen los mismos valores para todas las características. Cualquier categoría dada, por lo general, tendrá algunos estados que conducen a triunfos, algunos que conducen a empates, y algunos que conducen a pérdidas. La función de evaluación no sabe cuál es cada estado, pero sí puede devolver un valor que refleje la *proporción* de estados con cada resultado. Por ejemplo, supongamos que nuestra experiencia sugiere que el 72 por ciento de los estados encontrados en la categoría conduce a un triunfo (utilidad $+1$);

el 20 por ciento a una pérdida (-1), y el 8 por ciento a un empate (0). Entonces una evaluación razonable de estados en la categoría es el valor medio ponderado o **valor esperado**: $(0{,}72 \times +1) + (0{,}20 \times -1) + (0{,}08 \times 0) = 0{,}52$. En principio, el valor esperado se puede determinar para cada categoría, produciendo una función de evaluación que trabaja para cualquier estado. Mientras que con los estados terminales, la función de evaluación no tiene que devolver valores actuales esperados, la *ordenación* de los estados es el mismo.

VALOR ESPERADO

En la práctica, esta clase de análisis requiere demasiadas categorías y demasiada experiencia para estimar todas las probabilidades de ganar. En cambio, la mayoría de las funciones de evaluación calculan las contribuciones numéricas de cada característica y luego las *combinan* para encontrar el valor total. Por ejemplo, los libros de ajedrez dan, de forma aproximada, el **valor material** para cada pieza: cada peón vale 1, un caballo o el alfil valen 3, una torre 5, y la reina vale 9. Otras características como «la estructura buena del peón» y «seguridad del rey» podrían valer la mitad de un peón, por ejemplo. Estos valores de características, entonces, simplemente se suman para obtener la evaluación de la posición. Una ventaja segura equivalente a un peón da una probabilidad sustancial de ganar, y una ventaja segura equivalente a tres peones debería dar la casi victoria, como se ilustra en la Figura 6.8(a). Matemáticamente, a esta clase de función de evaluación se le llama **función ponderada lineal**, porque puede expresarse como

VALOR MATERIAL

FUNCIÓN PONDERADA LINEAL

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s),$$

donde cada w_i es un peso y cada f_i es una característica de la posición. Para el ajedrez, los f_i podrían ser los números de cada clase de piezas sobre el tablero, y los w_i podrían ser los valores de las piezas (1 para el peón, 3 para el alfil, etc.).

(a) Blanca mueve (b) Blanca mueve

Figura 6.8 Dos posiciones ligeramente diferentes del ajedrez. En (a), negro tiene una ventaja de un caballo y dos peones y ganará el juego. En (b), negro perderá después de que blanca capture la reina.

La suma de los valores de las características parece razonable, pero de hecho implica un axioma muy fuerte: que la contribución de cada característica sea *independiente* de los valores de las otras características. Por ejemplo, la asignación del valor 3 a un alfil no utiliza el hecho de que el alfil es más poderoso en la fase final, cuando tienen mucho espacio para maniobrar. Por esta razón, los programas actuales para el ajedrez, y otros juegos, también utilizan combinaciones *no lineales* de características. Por ejemplo, un par de alfiles podría merecer más la pena que dos veces el valor de un alfil, y un alfil merece más la pena en la fase final que al principio.

¡El lector habrá notado que las características y los pesos *no* son parte de las reglas del ajedrez! Provienen, desde hace siglos, de la experiencia humana al jugar al ajedrez. Considerando la forma lineal de evaluación, las características y los pesos dan como resultado la mejor aproximación a la ordenación exacta de los estados según su valor. En particular, la experiencia sugiere que una ventaja de más de un punto probablemente gane el juego, si no intervienen otros factores; una ventaja de tres puntos es suficiente para una victoria. En juegos donde no está disponible esta clase de experiencia, los pesos de la función de evaluación pueden estimarse con las técnicas de aprendizaje del Capítulo 18. La aplicación de estas técnicas al ajedrez han confirmado que un alfil es, en efecto, como aproximadamente tres peones.

Corte de la búsqueda

El siguiente paso es modificar la BUSQUEDA-ALFA-BETA de modo que llame a la función heurística EVAL cuando se corte la búsqueda. En términos de implementación, sustituimos las dos líneas de la Figura 6.7, que mencionan al TEST-TERMINAL, con la línea siguiente:

si TEST-CORTE(*estado, profundidad*) **entonces devolver** EVAL(*estado*)

También debemos llevar la contabilidad de la *profundidad* de modo que la profundidad actual se incremente sobre cada llamada recursiva. La aproximación más sencilla para controlar la cantidad de búsqueda es poner un límite de profundidad fijo, de modo que TEST-CORTE(*estado, profundidad*) devuelva verdadero para toda profundidad mayor que alguna profundidad fija *d*. (También debe devolver verdadero para todos los estados terminales, tal como hizo el TEST-TERMINAL.) La profundidad *d* se elige de modo que la cantidad de tiempo usado no exceda de lo que permiten las reglas del juego.

Una aproximación más robusta es aplicar profundidad iterativa, como se definió en el Capítulo 3. Cuando el tiempo se agota, el programa devuelve el movimiento seleccionado por la búsqueda completa más profunda. Sin embargo, estas aproximaciones pueden conducir a errores debido a la naturaleza aproximada de la función de evaluación. Considere otra vez la función de evaluación simple para el ajedrez basada en la ventaja del material. Suponga los programas de búsquedas al límite de profundidad, alcanzando la posición de la Figura 6.8(b), donde Negro aventaja por un caballo y dos peones. Esto estaría representado por el valor heurístico del estado, declarando que el estado conducirá probablemente a un triunfo de Negro. Pero en el siguiente movimiento de Blanco captura a la reina de Negro. De ahí, que la posición es realmente ganadora para Blanco, y que puede verse mirando una capa más.

ESTABLE

Obviamente, se necesita un test del límite más sofisticado. La función de evaluación debería aplicarse sólo a posiciones que son **estables** (es decir, improbablemente expuestas a grandes oscilaciones en su valor en un futuro próximo). En el ajedrez, por ejemplo, las posiciones en las cuales se pueden hacer capturas no son estables para una función de evaluación que solamente cuenta el material. Las posiciones no estables se pueden extender hasta que se alcancen posiciones estables. A esta búsqueda suplementaria se le llama **búsqueda de estabilidad o de reposo**; a veces se restringe a sólo ciertos tipos de movimientos, como movimientos de captura, que resolverán rápidamente la incertidumbre en la posición.

BÚQUEDA DE ESTABILIDAD

EFECTO HORIZONTE

El **efecto horizonte** es más difícil de eliminar. Se produce cuando el programa afronta un movimiento, del oponente, que causa un daño serio e inevitable. Considere el juego de ajedrez de la Figura 6.9. Negro aventaja en el material, pero si Blanco puede avanzar su peón de la séptima fila a la octava, el peón se convertirá en una reina y creará un triunfo fácil para Blanco. Negro puede prevenir este resultado en la capa 14 dando jaque a la reina blanca con la torre, pero al final, inevitablemente el peón se convertirá en una reina. El problema con la búsqueda de profundidad fija es que se cree que esquivar estos movimientos evitan el movimiento de convertirse en reina, decimos que los movimientos de esquivar empujan el movimiento de convertirse en reina (inevitable) «sobre el horizonte de búsqueda» a un lugar donde no puede detectarse.

EXTENSIONES EXCEPCIONALES

Cuando las mejoras de *hardware* nos lleven a realizar búsquedas más profundas, se espera que el efecto horizonte ocurra con menos frecuencia (las secuencias que tardan mucho tiempo son bastante raras). El uso de **extensiones excepcionales** también ha sido bastante eficaz para evitar el efecto horizonte sin añadir demasiado coste a la búsqueda. Una extensión excepcional es un movimiento que es «claramente mejor» que todos los demás en una posición dada. Una búsqueda de extensión excepcional puede ir más allá del límite de profundidad normal sin incurrir mucho en el coste porque su factor de

Negra mueve

Figura 6.9 El efecto horizonte. Una serie de jaques de la torre negra fuerza al movimiento de convertirse en reina (inevitable) de Blanco «sobre el horizonte» y hace que esta posición parezca un triunfo para Negro, cuando realmente es un triunfo para Blanco.

ramificación es 1. (Se puede pensar que la búsqueda de estabilidad es una variante de extensiones excepcionales.) En la Figura 6.9, una búsqueda de extensión excepcional encontrará el movimiento de convertirse en reina, a condición de que los movimientos de jaque de Negro y los movimientos del rey blanco puedan identificarse como «claramente mejores» que las alternativas.

Hasta ahora hemos hablado del corte de la búsqueda a un cierto nivel y que la poda alfa-beta, probablemente, no tiene ningún efecto sobre el resultado. Es también posible hacer la **poda hacia delante**, en la que podamos inmediatamente algunos movimientos de un nodo. Claramente, la mayoría de la gente que juega al ajedrez sólo considera unos pocos movimientos de cada posición (al menos conscientemente). Lamentablemente, esta aproximación es bastante peligrosa porque no hay ninguna garantía de que el mejor movimiento no sea podado. Esto puede ser desastroso aplicado cerca de la raíz, porque entonces a menudo el programa omitirá algunos movimientos «evidentes». La poda hacia delante puede usarse en situaciones especiales (por ejemplo, cuando dos movimientos son simétricos o equivalentes, sólo consideramos uno) o para nodos profundos en el árbol de búsqueda.

PODA HACIA DELANTE

La combinación de todas las técnicas descritas proporciona un programa que puede jugar al ajedrez loablemente (o a otros juegos). Asumamos que hemos implementado una función de evaluación para el ajedrez, un test del límite razonable con una búsqueda de estabilidad, y una tabla de transposiciones grande. También asumamos que, después de meses de intentos tediosos, podemos generar y evaluar alrededor de un millón de nodos por segundo sobre los últimos PCs, permitiéndonos buscar aproximadamente 200 millones de nodos por movimiento bajo un control estándar del tiempo (tres minutos por movimiento). El factor de ramificación para el ajedrez es aproximadamente 35, como media, y 35^5 son aproximadamente 50 millones, así si usamos la búsqueda minimax podríamos mirar sólo cinco capas. Aunque no sea incompetente, tal programa puede ser engañado fácilmente por un jugador medio de ajedrez, el cual puede planear, de vez en cuando, seis u ocho capas. Con la búsqueda alfa-beta nos ponemos en aproximadamente 10 capas, y resulta un nivel experto del juego. La Sección 6.7 describe técnicas de poda adicionales que pueden ampliar la profundidad eficaz de búsqueda a aproximadamente 14 capas. Para alcanzar el nivel de gran maestro necesitaríamos una función de evaluación ajustada y una base de datos grande de movimientos de apertura y movimientos finales óptimos. No sería malo tener un supercomputador para controlar este programa.

6.5 Juegos que incluyen un elemento de posibilidad

En la vida real, hay muchos acontecimientos imprevisibles externos que nos ponen en situaciones inesperadas. Muchos juegos reflejan esta imprevisibilidad con la inclusión de un elemento aleatorio, como el lanzamiento de dados. De esta manera, ellos nos dan un paso más cercano a la realidad, y vale la pena ver cómo afecta al proceso de toma de decisiones.

Backgammon es un juego típico que combina la suerte y la habilidad. Se hacen rodar unos dados, al comienzo del turno de un jugador, para determinar los movimientos

legales. En la posición *backgammon* de la Figura 6.10, por ejemplo, Blanco ha hecho rodar un 6-5, y tiene cuatro movimientos posibles.

Aunque Blanco sabe cuáles son sus propios movimientos legales, no sabe lo que le va a salir a Negro con los dados y por eso no sabe cuáles serán sus movimientos legales. Esto significa que Blanco no puede construir un árbol de juegos estándar de la forma que vimos en el ajedrez y tic-tac-toe. Un árbol de juegos en el *backgammon* debe incluir **nodos de posibilidad** además de los nodos MAX y MIN. En la Figura 6.11 se rodean con círculos los nodos de posibilidad. Las ramas que salen desde cada nodo posibilidad denotan las posibles tiradas, y cada una se etiqueta con la tirada y la posibilidad de que ocurra. Hay 36 resultados al hacer rodar dos dados, cada uno igualmente probable; pero como un 6-5 es lo mismo que un 5-6, hay sólo 21 resultado distintos. Los seis dobles (1-1 a 6-6) tienen una posibilidad de 1/36, los otros 15 resultados distintos un 1/18 cada uno.

NODOS DE POSIBILIDAD

El siguiente paso es entender cómo se toman las decisiones correctas. Obviamente, todavía queremos escoger el movimiento que conduzca a la mejor posición. Sin embargo, las posiciones no tienen valores minimax definidos. En cambio, podemos calcular el **valor esperado**, donde la expectativa se toma sobre todos los posibles resultados que podrían ocurrir. Éste nos conduce a generalizar el **valor minimax** para juegos deterministas a un **valor minimaxesperado** para juegos con nodos de posibilidad. Los nodos terminales y los MAX y MIN (para los que se conocen sus resultados) trabajan exactamente del mismo modo que antes; los nodos de posibilidad se evalúan tomando

VALOR MINIMAXESPERADO

Figura 6.10 Una posición típica del *backgammon*. El objetivo del juego es mover todas las fichas del tablero. Blanco mueve a la derecha hacia 25, y los movimientos de Negro al contrario, hacia 0. Una ficha puede moverse a cualquier posición a menos que haya varias piezas del oponente; si hay un oponente, es capturado y debe comenzar. En la posición mostrada, Blanca ha sacado 6-5 y debe elegir entre cuatro movimientos legales: (5-10,5-11), (5-11,19-24), (5-10,10-16) y (5-11,11-16).

Figura 6.11 Árbol de juegos esquemático para una posición del *backgammon*.

el promedio ponderado de los valores que se obtienen de todos los resultados posibles, es decir,

$$\text{MINIMAXESPERADO}(n) = \begin{cases} \text{UTILIDAD}(n) & \text{si } n \text{ es un nodo terminal} \\ \max_{s \in Sucesores(n)} \text{MINIMAXESPERADO}(s) & \text{si } n \text{ es un nodo MAX} \\ \min_{s \in Sucesores(n)} \text{MINIMAXESPERADO}(s) & \text{si } n \text{ es un nodo MIN} \\ \sum_{s \in Sucesores(n)} P(s) \cdot \text{MINIMAXESPERADO}(s) & \text{si } n \text{ es un nodo posibilidad} \end{cases}$$

donde la función sucesor para un nodo de posibilidad n simplemente aumenta el estado n con cada resultado posible para producir cada sucesor s, y $P(s)$ es la probabilidad de que ocurra ese resultado. Estas ecuaciones pueden aplicarse recursivamente hasta la raíz del árbol, como en minimax. Dejamos los detalles del algoritmo como un ejercicio.

Evaluación de la posición en juegos con nodos de posibilidad

Como con minimax, la aproximación obvia es cortar la búsqueda en algún punto y aplicar una función de evaluación a cada hoja. Uno podría pensar que las funciones de evaluación para juegos como *backgammon* deberían ser como funciones de evaluación para el ajedrez (solamente tienen que dar tanteos más altos a mejores posiciones). Pero de hecho, la presencia de nodos de posibilidades significa que uno tiene que tener más cui-

dado sobre la evaluación de valores medios. La Figura 6.12 muestra lo que sucede: con una función de evaluación que asigne valores [1,2,3,4] a las hojas, el movimiento A_1 es el mejor; con valores [1,20,30,400], el movimiento A_2 es el mejor. ¡De ahí, que los programas se comportan de forma totalmente diferente si hacemos un cambio de escala de algunos valores de la evaluación! Resulta que, para evitar esta sensibilidad, la función de evaluación debe ser una transformación *lineal positiva* de la probabilidad de ganancia desde una posición (o, más generalmente, de la utilidad esperada de la posición). Esto es una propiedad importante y general de situaciones en las que está implicada la incertidumbre, y hablaremos de ello en el Capítulo 16.

Complejidad del minimaxesperado

Si el programa supiera de antemano todos los resultados de las tiradas que ocurrirían para el resto del juego, resolver un juego con dados sería como resolver un juego sin dados, en el que minimax lo hace en $O(b^m)$ veces. Como minimaxesperado considera también todas las secuencias de las tiradas posibles, tendrá $O(b^m n^m)$, donde n es el número de resultados distintos.

Incluso si la profundidad de búsqueda se limita a una pequeña profundidad d, el coste adicional, comparado con el de minimax, lo hace poco realista para considerar anticiparse muy lejos en la mayoría de los juegos de azar. En *backgammon* n es 21 y b está por lo general alrededor de 20, pero en algunas situaciones podemos llegar a 4.000 para los resultados dobles. Tres capas serían, probablemente, todo lo que podríamos manejar.

Otro modo de ver el problema es este: la ventaja de alfa-beta consiste en ignorar los progresos futuros que apenas van a suceder, dado el mejor juego. Así, se concentra en acontecimientos probables. En juegos con dados, *no* hay secuencias probables de movimientos, porque para que ocurran esos movimientos, los dados tendrían que salir primero del modo correcto para hacerlos legales. Este es un problema general siempre que aparece la incertidumbre: las posibilidades se multiplican enormemente, y la formación

Figura 6.12 Una transformación que conserva el orden sobre los valores de las hojas cambia el mejor movimiento.

de proyectos detallados de acciones se hacen inútiles, porque el mundo probablemente no jugará así.

Sin duda esto le habrá ocurrido al lector que haya aplicado algo como la poda alfa-beta a árboles de juegos con nodos posibilidad. Resulta que esto podría ocurrir. El análisis para los nodos MIN y MAX no se altera, pero sí podemos podar también los nodos de posibilidades, usando un poco de ingenio. Consideremos el nodo de posibilidad *C* de la Figura 6.11 y lo que le pasa a su valor cuando examinamos y evaluamos a sus hijos. ¿Es posible encontrar un límite superior sobre el valor de *C* antes de que hayamos mirado a todos sus hijos? (Recuerde que esto es lo que alfa-beta necesita para podar un nodo y su subárbol.) A primera vista, podría parecer imposible, porque el valor de *C* es el *promedio* de los valores de sus hijos. Hasta que no hayamos visto todos los resultados de las tiradas, este promedio podría ser cualquier cosa, porque los hijos no examinados podrían tener cualquier valor. Pero si ponemos límites sobre los valores posibles de la función de utilidad, entonces podemos llegar a poner límites para el promedio. Por ejemplo, si decimos que todos los valores de utilidad están entre +3 y-3, entonces el valor de los nodos hoja está acotado, y por su parte *podemos* colocar una cota superior sobre el valor de un nodo de posibilidad sin ver a todos sus hijos.

Juegos de cartas

Los juegos de cartas son interesantes por muchos motivos además de su conexión con los juegos de azar. Entre la enorme variedad de juegos, nos centraremos en aquellos en los cuales las cartas se reparten al azar al principio del juego, y cada jugador recibe un conjunto de cartas que no son visibles a los otros jugadores. Tales juegos incluyen bridge, *whist*, corazones y algunas formas del póker.

A primera vista, podría parecer que los juegos de cartas son como juegos de dados: ¡las cartas se reparten al azar y determinan los movimientos disponibles para cada jugador, como si todos los dados fueran lanzados al principio! Perseguiremos esta observación más tarde. Resultará ser bastante útil en la práctica. También se equivoca bastante, por motivos interesantes.

Imaginemos dos jugadores, MAX y MIN, jugando algunas manos de bridge con cuatro cartas visibles. Las manos son, donde MAX juega primero:

MAX: ♡6 ♢6 ♣9 8 MIN: ♡4 ♠2 ♣10 5

Suponga que MAX sale con ♣9, MIN debe seguir ahora el palo, jugando el ♣10 o el ♣5. MIN juega el ♣10 y gana la mano. MIN va después y sale con ♠2. MAX no tiene picas (y así no puede ganar la mano) y por lo tanto debe tirar alguna carta. La opción obvia es ♢6 porque las otras dos cartas restantes son ganadoras. Ahora, cualquier carta que MIN saque para la siguiente mano, MAX ganará (ganará ambas manos) y el juego será de empate con dos manos cada uno. Es fácil mostrar, usando una variante conveniente de minimax (Ejercicio 6.12), que la salida de MAX con el ♣9 es de hecho una opción óptima.

Ahora vamos a modificar la mano de MIN, sustituyendo los ♡4 con los ♢4:

MAX: ♡6 ♢6 ♣9 8 MIN: ♡4 ♠2 ♣10 5

Los dos casos son completamente simétricos: el juego será idéntico, salvo que en la segunda mano MAX tirará el ♡6. Otra vez, el juego será de empate a dos manos cada uno y la salida con ♣9 es una opción óptima.

Hasta ahora, bien. Ahora vamos a esconder una de las cartas de MIN: MAX sabe que MIN tiene lo de la primera mano (con el ♡4) o lo de la segunda (con el ♢4), pero no sabe cuál tiene realmente. MAX razona como sigue:

> El ♣9 es una opción óptima contra la primera mano de MIN y contra la segunda mano de MIN, entonces debe ser óptima ahora porque sé que MIN tiene una de las dos manos.

Más generalmente, MAX usa lo que podríamos llamar «hacer un promedio sobre la clarividencia». La idea es evaluar una línea de acción, cuando hay cartas no visibles, calculando primero el valor de minimax de esa acción para cada reparto posible de cartas, y luego calcular el valor esperado usando la probabilidad de cada reparto.

Si piensa que esto es razonable (o si no tiene ni idea porque no entiende el bridge), considere la siguiente historia:

> Día 1: el Camino A conduce a un montón de piezas de oro; el camino B conduce a una bifurcación. Tome hacia la izquierda y encontrará un montón de joyas, pero tome hacia la derecha y será atropellado por un autobús.
> Día 2: el Camino A conduce a un montón de piezas de oro; el camino B conduce a una bifurcación. Tome hacia la derecha y encontrará un montón de joyas, pero tome hacia la izquierda y será atropellado por un autobús.
> Día 3: el camino A conduce a un montón de piezas de oro; el camino B conduce a una bifurcación. Adivine correctamente y encontrará un montón de joyas, pero adivine incorrectamente y será atropellado por un autobús.

Obviamente, no es irrazonable tomar el camino B durante los dos primeros días. Ninguna persona en su sano juicio, sin embargo, tomaría el camino B durante el Día 3. Esto es exactamente lo que sugiere un promedio sobre la clarividencia: el camino B es óptimo en las situaciones de Día 1 y Día 2; por lo tanto es óptimo durante el Día 3, porque una de las dos situaciones anteriores debe conseguirse. Volvamos al juego de cartas: después de que MAX salga con el ♣9, MIN gana con el ♣10. Como antes, MIN sale con el ♠2, y ahora MAX está como en la bifurcación del camino sin ninguna instrucción. Si MAX tira el ♡6 y MIN todavía tiene ♡4, el ♡4 se hace ganador y MAX pierde el juego. Del mismo modo, si MAX tira ♢6 y MIN todavía tiene el ♢4, MAX también pierde. Por lo tanto, jugar primero el ♣9 conduce a una situación donde MAX tiene una posibilidad del 50 por ciento de perder. (Sería mucho mejor jugar primero el ♡6 y el ♢6, garantizando un empate.)

La lección que podemos sacar de todo esto es que cuando falla la información, hay que considerar *la información que se tendrá* en cada punto del juego. El problema con el algoritmo de MAX es que asume que en cada posible reparto, el juego procederá *como si todas las cartas fueran visibles*. Como muestra nuestro ejemplo, esto conduce a MAX a actuar como si toda la incertidumbre *futura* vaya a ser resuelta cuando aparezca. El algoritmo de MAX decidirá también no reunir la información (ni *proporcionar* información a un compañero), porque en cada reparto no hay ninguna necesidad de hacerlo; sin embargo en juegos como el bridge, es a menudo una buena idea jugar una carta que ayudará a descubrir cosas sobre las cartas del adversario o informar a su compañero sobre

sus propias cartas. Estas clases de comportamientos se generan automáticamente por un algoritmo óptimo para juegos de la información imperfecta. Tal algoritmo no busca en el espacio de estados del mundo (las manos de las cartas), sino en el espacio de **estados de creencia** (creencia de quién tiene qué cartas, con qué probabilidades). Seremos capaces de explicar el algoritmo correctamente en el Capítulo 17, una vez que hayamos desarrollado la maquinaria probabilística necesaria. En ese capítulo, ampliaremos también un punto final y muy importante: en juegos de información imperfecta, lo mejor es dar tan poca información al oponente como sea posible, y a menudo el mejor modo de hacerlo es actuar de manera *impredecible*. Por eso, los inspectores de sanidad hacen visitas de inspección aleatorias a los restaurantes.

6.6 Programas de juegos

Podríamos decir que jugar a juegos es a IA como el Gran Premio de carreras de automóviles es a la industria de coches: los programas de juegos son deslumbrantemente rápidos, máquinas increíblemente bien ajustadas que incorporan técnicas muy avanzadas de la ingeniería, pero no son de mucho uso para hacer la compra. Aunque algunos investigadores crean que jugar a juegos es algo irrelevante en la corriente principal de IA, se sigue generando entusiasmo y una corriente estable de innovaciones que se han adoptado por la comunidad.

AJEDREZ

Ajedrez: en 1957, Herbert Simon predijo que dentro de 10 años los computadores ganarían al campeón mundial humano. Cuarenta años más tarde, el programa Deep Blue derrotó a Garry Kasparov en un partido de exhibición a seis juegos. Simon se equivocó, pero sólo por un factor de 4. Kasparov escribió:

> El juego decisivo del partido fue el juego 2, que dejó una huella en mi memoria... Vimos algo que fue más allá de nuestras expectativas más salvajes de cómo un computador sería capaz de prever las consecuencias posicionales a largo plazo de sus decisiones. La máquina rechazó moverse a una posición que tenía una ventaja decisiva a corto plazo, mostrando un sentido muy humano del peligro. (Kasparov, 1997)

Deep Blue fue desarrollado por Murray Campbell, Feng-Hsiung Hsu, y Joseph Hoane en IBM (*véase* Campbell *et al*., 2002), construido sobre el diseño de Deep Thought desarrollado anteriomente por Campbell y Hsu en Carnegie Mellon. La máquina ganadora era un computador paralelo con 30 procesadores IBM RS/6000 que controlaba la «búsqueda *software*» y 480 procesadores VLSI, encargados para el ajedrez, que realizaron la generación de movimientos (incluso el movimiento de ordenación), la «búsqueda *hardware*» para los últimos niveles del árbol, y la evaluación de los nodos hoja. Deep Blue buscó 126 millones de nodos por segundo, como regla general, con una velocidad máxima de 330 millones de nodos por segundo. Generó hasta 30 billones de posiciones por movimiento, y alcanzó la profundidad 14 rutinariamente. El corazón de la máquina es una búsqueda alfa-beta estándar de profundidad iterativa con una tabla de transposiciones, pero la llave de su éxito parece haber estado en su capacidad de generar extensiones más allá del límite de profundidad para líneas suficientemente interesantes. En

algunos casos la búsqueda alcanzó una profundidad de 40 capas. La función de evaluación tenía más de 8.000 características, muchas de ellas describiendo modelos muy específicos de las piezas. Se usó una «salida de libro» de aproximadamente 4.000 posiciones, así como una base de datos de 700.000 jugadas de gran maestro de las cuales se podrían extraer algunas recomendaciones de consenso. El sistema también utilizó una gran base de datos de finales del juego de posiciones resueltas, conteniendo todas las posiciones con cinco piezas y muchas con seis piezas. Esta base de datos tiene el considerable efecto de ampliar la profundidad efectiva de búsqueda, permitiendo a Deep Blue jugar perfectamente en algunos casos aun cuando se aleja del jaque mate.

El éxito de Deep Blue reforzó la creencia de que el progreso en los juegos de computador ha venido principalmente del *hardware* cada vez más poderoso (animado por IBM). Los creadores de Deep Blue, por otra parte, declaran que las extensiones de búsqueda y la función de evaluación eran también críticas (Campbell *et al*., 2002). Además, sabemos que mejoras algorítmicas recientes han permitido a programas, que se ejecutan sobre computadores personales, ganar cada Campeonato Mundial de Ajedrez de computadores desde 1992, a menudo derrotando a adversarios masivamente paralelos que podrían buscar 1.000 veces más nodos. Una variedad de poda heurística se utiliza para reducir el factor de ramificación efectivo a menos de 3 (comparado con el factor de ramificación actual de aproximadamente 35). Lo más importante de ésta es la heurística de **movimiento nulo**, que genera una cota inferior buena sobre el valor de una posición, usando una búsqueda superficial en la cual el adversario consigue moverse dos veces. Esta cota inferior a menudo permite la poda alfa-beta sin el costo de una búsqueda de profundidad completa. También es importante la **poda de inutilidad**, la cual ayuda ha decidir de antemano qué movimientos causarán un corte beta en los nodos sucesores.

MOVIMIENTO NULO

PODA DE INUTILIDAD

El equipo de Deep Blue rehusó la posibilidad de un nuevo partido con Kasparov. En cambio, la competición principal más reciente de 2002 destacó el programa FRITZ contra el campeón mundial Vladimir Kramnik. El partido a ocho juegos terminó en un empate. Las condiciones del partido eran mucho más favorables al humano, y el *hardware* era un computador personal ordinario, no un supercomputador. De todos modos, Kramnik comentó que «está ahora claro que el programa y el campeón mundial son aproximadamente iguales».

DAMAS

Damas: en 1952, Arthur Samuel de IBM, trabajando en sus ratos libres, desarrolló un programa de damas que aprendió su propia función de evaluación jugando con él mismo miles de veces. Describimos esta idea más detalladamente en el Capítulo 21. El programa de Samuel comenzó como un principiante, pero después, sólo unos días de auto-jugar, se había mejorado más allá del propio nivel de Samuel (aunque él no fuera un jugador fuerte). En 1962 derrotó a Robert Nealy, un campeón en «damas ciegas», por un error por su parte. Muchas personas señalaron que, en damas, los computadores eran superiores a la gente, pero no era la cuestión. De todos modos, cuando uno considera que el equipo calculador de Samuel (un IBM 704) tenía 10.000 palabras de memoria principal, cinta magnetofónica para el almacenaje a largo plazo y un procesador de ,000001 GHz, el triunfo sigue siendo un gran logro.

Pocas personas intentaron hacerlo mejor hasta que Jonathan Schaeffer y colegas desarrollaran Chinook, que se ejecuta sobre computadores personales y usa la búsqueda alfa-

beta. Chinook usa una base de datos precalculada de 444 billones de posiciones con ocho o menos piezas sobre el tablero para así hacer la fase final del juego de forma impecable. Chinook quedó segundo en el Abierto de Estados Unidos de 1990 y ganó el derecho a participar en el campeonato mundial. Entonces se ejecutó contra un problema, en la forma de Marion Tinsley. Dr. Tinsley había sido el campeón mundial durante más de 40 años, y había perdido sólo tres juegos en todo ese tiempo. En el primer partido contra Chinook, Tinsley sufrió sus cuartas y quintas derrotas, pero ganó el partido 20.5-18.5. El partido del campeonato mundial en agosto de 1994 entre Tinsley y Chinook se terminó prematuramente cuando Tinsley tuvo que retirarse por motivos de salud. Chinook se convirtió en el campeón mundial oficial.

Schaeffer cree que, con bastante poder calculador, la base de datos de fases finales podría ampliarse hasta el punto donde una búsqueda hacia delante desde la posición inicial alcanzaría siempre posiciones resueltas, es decir, las damas estarían completamente resueltas. (Chinook ha anunciado un triunfo tan sólo en cinco movimientos.) Esta clase de análisis exhaustivo puede hacerse a mano para tic-tac-toe 3x3 y se ha hecho con el computador para Qubic (tic-tac-toe 4x4x4), Go-Moku (cinco en fila), y Morris de nueve-hombres (Gasser, 1998). El trabajo notable de Ken Thompson y Lewis Stiller (1992) resolvió todo el ajedrez con cinco piezas y las fases finales de seis piezas, poniéndolas a disposición sobre Internet. Stiller descubrió un caso donde existía un jaque mate forzado, pero requirió 262 movimientos; esto causó alguna consternación porque las reglas del ajedrez requieren que ocurra algún «progreso» en 50 movimientos.

OTELO

Otelo: también llamado Reversi, es probablemente más popular como un juego de computador que como un juego de mesa. Tiene un espacio de búsqueda más pequeño que el ajedrez, por lo general de cinco a 15 movimientos legales, pero desde los comienzos se tuvo que desarrollar la evaluación experta. En 1997, el programa Logistello (Buro, 2002) derrotó al campeón mundial humano, Takeshi Murakami, por seis juegos a ninguno. Se reconoce, generalmente, que la gente no es igual a los computadores en Otelo.

BACKGAMMON

Backgammon: la Sección 6.5 explicó por qué la inclusión de incertidumbre, provocada por el lanzamiento de los dados, hace de la búsqueda un lujo costoso. La mayor parte de los trabajos sobre *backgammon* se han centrado en la mejora de la función de evaluación. Gerry Tesauro (1992) combinó el aprendizaje por refuerzo de Samuel con técnicas de redes neuronales (Capítulo 20) para desarrollar un evaluador notablemente exacto usado con una búsqueda a profundidad 2 o 3. Después de jugar más de un millón de juegos de entrenamiento contra él mismo, el programa de Tesauro, TD-GAMMON, se sitúa, seguramente, entre los tres primeros jugadores del mundo. Las opiniones del programa sobre los movimientos de apertura del juego han alterado radicalmente la sabiduría recibida.

GO

Go: es el juego de mesa más popular de Asia, requiriendo al menos tanta disciplina de sus profesionales como el ajedrez. Como el tablero es de 19x19, el factor de ramificación comienza en 361, que desalienta también a los métodos regulares de búsqueda. Hasta 1997 no había ningún programa competente, pero ahora los programas a menudo juegan respetablemente. La mayor parte de los mejores programas combinan técnicas de reconocimiento de modelos (cuando aparece el siguiente modelo de piezas, este

movimiento debe considerarse) con la búsqueda limitada (decide si estas piezas pueden ser capturadas, y quedan dentro del área local). Los programas más fuertes, en el momento de estar escribiendo, son probablemente Goemate de Chen Zhixing y Go4++ de Michael Reiss, cada uno valorado en alrededor de 10 kyu (aficionado débil). Go es un área que probablemente se beneficiará de la investigación intensiva que utiliza métodos de razonamiento más sofisticados. El éxito puede venir de encontrar modos de integrar varias líneas del razonamiento local sobre cada uno de los muchos «subjuegos» ligeramente conectados en los que Go se puede descomponer. Tales técnicas serían de enorme valor para sistemas inteligentes en general.

BRIDGE

Bridge: es un juego de información imperfecta: las cartas de un jugador se esconden de los otros jugadores. El bridge es también un juego *multijugador* con cuatro jugadores en vez de dos, aunque los jugadores se emparejen en dos equipos. Cuando lo vimos en la Sección 6.5, el juego óptimo en el bridge puede incluir elementos de reunión de información, comunicación, tirarse un farol, y el ponderado cuidadoso de probabilidades. Muchas de estas técnicas se usan en el programa de Bridge Baron™ (Smith *et al*., 1998), que ganó el campeonato de bridge de computadores de 1997. Mientras no juega óptimamente, Bridge Baron es uno de los pocos sistemas de juegos en usar planes complejos y jerárquicos (*véase* el Capítulo 12) que implican ideas de alto nivel como **astucia** y **aprieto**, que son familiares para los jugadores de bridge.

El programa GIB (Ginsberg, 1999) ganó el campeonato 2000 con decisión. GIB usa el método de «hacer un promedio sobre la clarividencia», con dos modificaciones cruciales. Primero, antes de examinar cuán bien trabaja cada opción para cada plan posible de las cartas escondidas, (de las cuales puede ser hasta 10 millones) examina una muestra aleatoria de 100 planes. Segundo, GIB usa la **generalización basada en explicaciones** para calcular y guardar las reglas generales para el juego óptimo en varias clases estándar de situaciones. Esto permite resolver cada reparto *exactamente*. La exactitud táctica del GIB compensa su inhabilidad de razonar sobre la información. Terminó el 12º de 35 en la competición de igualdad (implicando solamente el juego de una mano) en el campeonato mundial humano de 1998, excediendo las expectativas de muchos expertos humanos.

6.7 Discusión

Como el cálculo de decisiones óptimas en juegos es intratable en la mayoría de los casos, todos los algoritmos deben hacer algunas suposiciones y aproximaciones. La aproximación estándar, basada en minimax, funciones de evaluación y alfa-beta, es solamente un modo de hacerlo. Probablemente porque la propusieron tan pronto, la aproximación estándar fue desarrollada intensivamente y domina otros métodos en los juegos de turnos. Algunos en el campo creen que esto ha causado que jugar a juegos llegue a divorciarse de la corriente principal de investigación de IA, porque la aproximación estándar no proporciona mucho más espacio para nuevas perspicacias en cuestiones generales de la toma de decisiones. En esta sección, vemos las alternativas.

Primero, consideremos minimax. Minimax selecciona un movimiento óptimo en un árbol de búsqueda *a condición de que las evaluaciones de los nodos hoja sean exactamente correctas*. En realidad, las evaluaciones son generalmente estimaciones rudimentarias del valor de una posición y se consideran que tienen asociados errores grandes. La Figura 6.13 muestra un árbol de juegos de dos capas para el cual minimax parece inadecuado. Minimax aconseja tomar la rama derecha, mientras que es bastante probable que el valor real de la rama izquierda sea más alto. La opción de minimax confía suponiendo que *todos* los nodos etiquetados con valores 100, 101, 102 y 100 sean *realmente* mejores que el nodo etiquetado con el valor 99. Sin embargo, el hecho de que el nodo etiquetado con 99 tiene hermanos etiquetados con 1.000 sugiere que, de hecho, podría tener un valor real más alto. Un modo de tratar con este problema es tener una evaluación que devuelva una *distribución de probabilidad* sobre valores posibles. Entonces uno puede calcular la distribución de probabilidad del valor del padre usando técnicas estadísticas. Lamentablemente, los valores de los nodos hermanos están, por lo general, muy correlacionados, por consiguiente puede ser de cálculo costoso, requisito importante para obtener la información.

Después, consideramos el algoritmo de búsqueda que genera el árbol. El objetivo de un diseñador de algoritmos es especificar un cálculo de ejecución rápida y que produzca un movimiento bueno. El problema más obvio con el algoritmo alfa-beta es que está diseñado, no solamente para seleccionar un movimiento bueno, sino también para calcular límites sobre los valores de todos los movimientos legales. Para ver por qué esta información suplementaria es innecesaria, consideremos una posición en la cual hay sólo un movimiento legal. La búsqueda alfa-beta todavía generará y evaluará un grande, y totalmente inútil, árbol de búsqueda. Desde luego, podemos insertar un test en el algoritmo, pero éste simplemente esconderá el problema subyacente: muchos de los cálculos hechos por alfa-beta son en gran parte irrelevantes. Tener sólo un movimiento legal no es mucho más diferente que tener varios movimientos legales, uno de los cuales es excelente y el resto obviamente desastroso. En una situación «favorita clara» como ésta, sería mejor alcanzar una decisión rápida después de una pequeña cantidad de búsqueda, que gastar el tiempo que podría ser más provechoso más tarde en una posición más problemática. Esto conduce a la idea de la *utilidad de una expansión de un nodo*. Un algoritmo de búsqueda bueno debería seleccionar expansiones de nodos de utilidad alta

Figura 6.13 Un árbol de juegos de dos capas para el cual minimax puede ser inadecuado.

(es decir que probablemente conducirán al descubrimiento de un movimiento considerablemente mejor). Si no hay ninguna expansión de nodos cuya utilidad sea más alta que su coste (en términos de tiempo), entonces el algoritmo debería dejar de buscar y hacer un movimiento. Notemos que esto funciona no solamente para situaciones favoritas claras, sino también para el caso de movimientos *simétricos*, para los cuales ninguna cantidad de la búsqueda mostrará que un movimiento es mejor que otro.

META-RAZONAMIENTO

A esta clase de razonamiento, sobre qué cálculos hacer, se le llama **meta-razonamiento** (razonamiento sobre el razonamiento). Esto se aplica no solamente a juegos, sino a cualquier clase del razonamiento. Todos los cálculos se hacen para tratar de alcanzar mejores decisiones, todos tienen gastos, y todos tienen alguna probabilidad de causar una cierta mejora de la calidad de decisión. Alfa-beta incorpora la clase más simple de meta-razonamiento, un teorema en el sentido de que ciertas ramas del árbol pueden ignorarse sin perder. Es posible hacerlo mucho mejor. En el Capítulo 16, veremos cómo estas ideas pueden hacerse más precisas e implementables.

Finalmente, reexaminemos la naturaleza de la búsqueda en sí mismo. Algoritmos para la búsqueda heurística y para juegos trabajan generando secuencias de estados concretos, comenzando desde el estado inicial y luego aplicando una función de evaluación. Claramente, así no es como juega la gente. En el ajedrez, uno a menudo tiene en mente un objetivo particular (por ejemplo, atrapar la reina del adversario) y puede usar este objetivo para generar *selectivamente* el plan plausible para conseguirlo. Esta clase de **razonamiento dirigido por objetivos** o **planificación** a veces elimina, totalmente, la búsqueda combinatoria. (*Véase* la Parte IV.) PARADISE de David Wilkins (1980) es el único programa que ha usado el razonamiento dirigido por objetivos con éxito en el ajedrez: era capaz de resolver algunos problemas de ajedrez que requieren una combinación de 18 movimientos. Aún no es nada fácil entender cómo *combinar* las dos clases de algoritmos en un sistema robusto y eficiente, aunque Bridge Baron pudiera ser un paso en la dirección correcta. Un sistema totalmente integrado sería un logro significativo, no solamente para la investigación de juegos, sino también para la investigación de IA en general, porque esto sería una buena base para un agente general inteligente.

6.8 Resumen

Hemos visto una variedad de juegos para entender qué significa jugar óptimamente y entender cómo jugar bien en la práctica. Las ideas más importantes son las siguientes:

- Un juego puede definirse por el **estado inicial** (como se establece en el tablero), las **acciones** legales en cada estado, un **test terminal** (que dice cuándo el juego está terminado), y una **función de utilidad** que se aplica a los estados terminales.
- En juegos de suma cero de dos jugadores con **información perfecta**, el algoritmo **minimax** puede seleccionar movimientos óptimos usando una enumeración primero en profundidad del árbol de juegos.
- El algoritmo de búsqueda **alfa-beta** calcula el mismo movimiento óptimo que el minimax, pero consigue una eficiencia mucho mayor, eliminando subárboles que son probablemente irrelevantes.

- Por lo general, no es factible considerar el árbol entero de juegos (hasta con alfa-beta), entonces tenemos que cortar la búsqueda en algún punto y aplicar una **función de evaluación** que dé una estimación de la utilidad de un estado.
- Los juegos de azar pueden manejarse con una extensión del algoritmo minimax que evalúa un **nodo de posibilidad** tomando la utilidad media de todos sus nodos hijos, ponderados por la probabilidad de cada hijo.
- El juego óptimo en juegos de **información imperfecta**, como el bridge, requiere el razonamiento sobre los **estados de creencia** actuales y futuros de cada jugador. Una aproximación simple puede obtenerse haciendo un promedio del valor de una acción sobre cada configuración posible de la información ausente.
- Los programas pueden equipararse o pueden ganar a los mejores jugadores humanos en damas, Otelo y *backgammon,* y están cercanos en bridge. Un programa ha ganado al campeón mundial de ajedrez en un partido de exhibición. Los programas permanecen en el nivel aficionado en Go.

Notas bibliográficas e históricas

La temprana historia de los juegos mecánicos se estropeó por numerosos fraudes. El más célebre de estos fue «El Turco» de Baron Wolfgang von Kempelen (1734-1804), un supuesto autómata que jugaba al ajedrez, que derrotó a Napoleón antes de ser expuesto como la caja de bromas de un mago que escondía a un humano experto en ajedrez (*véase* Levitt, 2000). Jugó desde 1769 hasta 1854. En 1846, Charles Babbage (quien había sido fascinado por el Turco) parece haber contribuido a la primera discusión seria de la viabilidad del computador de ajedrez y de damas (Morrison y Morrison, 1961). Él también diseñó, pero no construyó, una máquina con destino especial para jugar a tic-tac-toe. La primera máquina real de juegos fue construida alrededor de 1890 por el ingeniero español Leonardo Torres y Quevedo. Se especializó en el «RTR» (rey y torre contra el rey), la fase final de ajedrez, garantizando un triunfo con el rey y torre desde cualquier posición.

El algoritmo minimax se remonta a un trabajo publicado en 1912 de Ernst Zermelo, el que desarrolló la teoría moderna de conjuntos. El trabajo, lamentablemente, tenía varios errores y no describió minimax correctamente. Un fundamento sólido, para la teoría de juegos, fue desarrollado en el trabajo seminal de *Theory of Games and Economic Behaviour* (von Neumann y Morgenstern, 1944), que incluyó un análisis en el que mostraba que algunos juegos *requieren* estrategias aleatorizadas (o imprevisibles). *Véase* el Capítulo 17 para más información.

Muchas figuras influyentes de los comienzos de los computadores, quedaron intrigadas por la posibilidad de jugar al ajedrez con un computador. Konrad Zuse (1945), la primera persona que diseñó un computador programable, desarrolló ideas bastante detalladas sobre cómo se podría hacer esto. El libro influyente de Norbert Wiener (1948), *Cybernetics,* habló de un diseño posible para un programa de ajedrez, incluso las ideas de búsqueda minimax, límites de profundidad, y funciones de evaluación. Claude Shannon (1950) presentó los principios básicos de programas modernos de juegos con mucho más detalle que Wiener. Él introdujo la idea de la búsqueda de estabilidad y describió algunas ideas para la búsqueda del árbol de juegos selectiva (no exhaustiva). Slater

(1950) y los que comentaron su artículo también exploraron las posibilidades para el juego de ajedrez por computador. En particular, I. J. Good (1950) desarrolló la noción de estabilidad independientemente de Shannon.

En 1951, Alan Turing escribió el primer programa de computador capaz de jugar un juego completo de ajedrez (*véase* Turing *et al*., 1953). Pero el programa de Turing nunca se ejecutó sobre un computador; fue probado por simulación a mano contra un jugador muy débil humano, que lo derrotó. Mientras tanto D. G. Prinz (1952) escribió , y realmente ejecutó, un programa que resolvió problemas de ajedrez, aunque no jugara un juego completo. Alex Bernstein escribió el primer programa para jugar un juego completo de ajedrez estándar (Bernstein y Roberts, 1958; Bernstein *et al*., 1958)[3].

John McCarthy concibió la idea de la búsqueda alfa-beta en 1956, aunque él no lo publicara. El programa NSS de ajedrez (Newell *et al*., 1958) usó una versión simplificada de alfa-beta; y fue el primer programa de ajedrez en hacerlo así. Según Nilsson (1971), el programa de damas de Arthur Samuel (Samuel, 1959, 1967) también usó alfa-beta, aunque Samuel no lo mencionara en los informes publicados sobre el sistema. Los trabajos que describen alfa-beta fueron publicados a principios de 1960 (Hart y Edwards, 1961; Brudno, 1963; Slagle, 1963b). Una implementación completa de alfa-beta está descrita por Slagle y Dixon (1969) en un programa para juegos de Kalah. Alfa-beta fue también utilizada por el programa «Kotok-McCarthy» de ajedrez escrito por un estudiante de John McCarthy (Kotok, 1962). Knuth y Moore (1975) proporcionan una historia de alfa-beta, junto con una demostración de su exactitud y un análisis de complejidad en tiempo. Su análisis de alfa-beta con un orden de sucesores aleatorio mostró una complejidad asintótica de $O((b/log\ b)^d)$, que pareció bastante triste porque el factor de ramificación efectivo $b/log\ b$ no es mucho menor que b. Ellos, entonces, se dieron cuenta que la fórmula asintótica es exacta sólo para $b > 1000$ más o menos, mientras que a menudo se aplica un $O(b^{3d/4})$ a la variedad de factores de ramificación encontrados en los juegos actuales. Pearl (1982b) muestra que alfa-beta es asintóticamente óptima entre todos los algoritmos de búsqueda de árbol de juegos de profundidad fija.

El primer partido de ajedrez de computador presentó al programa Kotok-McCarthy y al programa «ITEP» escrito a mediados de 1960 en el Instituto de Moscú de Física Teórica y Experimental (Adelson-Velsky *et al*., 1970). Este partido intercontinental fue jugado por telégrafo. Se terminó con una victoria 3-1 para el programa ITEP en 1967. El primer programa de ajedrez que compitió con éxito con la gente fue MacHack 6 (Greenblatt *et al*., 1967). Su grado de aproximadamente 1.400 estaba bien sobre el nivel de principiante de 1.000, pero era bajo comparado con el grado 2.800 o más que habría sido necesario para satisfacer la predicción de 1957 de Herb Simon de que un programa de computador sería el campeón mundial de ajedrez en el plazo de 10 años (Simon y Newell, 1958).

Comenzando con el primer Campeonato Norteamericano ACM de Ajedrez de computador en 1970, el concurso entre programas de ajedrez se hizo serio. Los programas a principios de 1970 se hicieron sumamente complicados, con varias clases de trucos para eliminar algunas ramas de búsqueda, para generar movimientos plausibles, etcétera.

[3] Newell *et al*. (1958) mencionan un programa ruso, BESM, que puede haber precedido al programa de Bernstein.

En 1974, el primer Campeonato Mundial de Ajedrez de computador fue celebrado en Estocolmo y ganado por Kaissa (Adelson-Velsky *et al*, 1975), otro programa de ITEP. Kaissa utilizó la aproximación mucho más directa de la búsqueda alfa-beta exhaustiva combinada con la búsqueda de estabilidad. El dominio de esta aproximación fue confirmado por la victoria convincente de CHESS 4.6 en el Campeonato Mundial de 1977 de Ajedrez de computador. CHESS 4.6 examinó hasta 400.000 posiciones por movimiento y tenía un grado de 1.900.

Una versión posterior de MacHack, de Greenblatt 6, fue el primer programa de ajedrez ejecutado sobre un *hardware* de encargo diseñado expresamente para el ajedrez (Moussouris *et al*., 1979), pero el primer programa en conseguir éxito notable por el uso del *hardware* de encargo fue Belle (Condon y Thompson, 1982). El *hardware* de generación de movimientos y de la evaluación de la posición de Belle, le permitieó explorar varios millones de posiciones por movimiento. Belle consiguió un grado de 2.250, y se hizo el primer programa de nivel maestro. El sistema HITECH, también un computador con propósito especial, fue diseñado por el antiguo Campeón de Ajedrez de Correspondencia Mundial Hans Berliner y su estudiante Carl Ebeling en CMU para permitir el cálculo rápido de la función de evaluación (Ebeling, 1987; Berliner y Ebeling, 1989). Generando aproximadamente 10 millones de posiciones por movimiento, HITECH se hizo el campeón norteamericano de computador en 1985 y fue el primer programa en derrotar a un gran maestro humano en 1987. Deep Thought, que fue también desarrollado en CMU, fue más lejos en la dirección de la velocidad pura de búsqueda (Hsu *et al*., 1990). Consiguió un grado de 2.551 y fue el precursor de Deep Blue. El Premio de Fredkin, establecido en 1980, ofreció 5.000 dólares al primer programa en conseguir un grado de maestro, 10.000 dólares al primer programa en conseguir un grado FEUA (Federación de los Estados Unidos de Ajedrez) de 2.500 (cerca del nivel de gran maestro), y 100.000 dólares para el primer programa en derrotar al campeón humano mundial. El premio de 5.000 dólares fue reclamado por Belle en 1983, el premio de 10.000 dólares por Deep Thought en 1989, y el premio de 100.000 dólares por Deep Blue por su victoria sobre Garry Kasparov en 1997. Es importante recordar que el éxito de Deep Blue fue debido a mejoras algorítmicas y de *hardware* (Hsu, 1999; Campbell *et al*., 2002). Las técnicas como la heurística de movimiento nulo (Beal, 1990) han conducido a programas que son completamente selectivos en sus búsquedas. Los tres últimos Campeonatos Mundiales de Ajedrez de computador en 1992, 1995 y 1999 fueron ganados por programas que se ejecutan sobre computadores personales. Probablemente la mayor parte de la descripción completa de un programa moderno de ajedrez la proporciona Ernst Heinz (2000), cuyo programa DARKTHOUGHT fue el programa de computador no comercial de rango más alto de los campeonatos mundiales de 1999.

Varias tentativas se han hecho para vencer los problemas «de la aproximación estándar» perfilados en la Sección 6.7. El primer algoritmo selectivo de búsqueda con un poco de base teórica fue probablemente B* (Berlinés, 1979), que intenta mantener límites de intervalos sobre el valor posible de un nodo en el árbol de juegos, más que darle una estimación valorada por un punto. Los nodos hoja son seleccionados para expansión en una tentativa de refinar los límites del nivel superior hasta que un movimiento sea «claramente mejor». Palay (1985) amplía la idea de B* para usar distribuciones de probabilidad en lugar de intervalos. La búsqueda del número de conspiración

de David McAllester (1988) expande los nodos hoja que, cambiando sus valores, podrían hacer que el programa prefiriera un nuevo movimiento en la raíz. MGSS* (Russell y Wefald, 1989) usa las técnicas teóricas de decisión del Capítulo 16 para estimar el valor de expansión de cada hoja en términos de mejora esperada de la calidad de decisión en la raíz. Jugó mejor que un algoritmo alfa-beta, en Otelo, a pesar de la búsqueda de un orden de magnitud de menos nodos. La aproximación MGSS* es, en principio, aplicable al control de cualquier forma de deliberación.

La búsqueda alfa-beta es, desde muchos puntos de vista, el análogo de dos jugadores al ramificar y acotar primero en profundidad, dominada por A* en el caso de agente simple. El algoritmo SSS* (Stockman, 1979) puede verse como A* de dos jugadores y nunca expande más nodos que alfa-beta para alcanzar la misma decisión. Las exigencias de memoria y los costos indirectos computacionales de la cola hacen que SSS* sea poco práctico en su forma original, pero se ha desarrollado una versión de espacio-lineal a partir del algoritmo RBFS (Korf y Chickering, 1996). Plaat *et al*. (1996) desarrollaron una nueva visión de SSS* como una combinación de alfa-beta y tablas de transposiciones, mostrando cómo vencer los inconvenientes del algoritmo original y desarrollando una nueva variante llamada MTD(*f*) que ha sido adoptada por varios programas superiores.

D. F. Beal (1980) y Dana Nau (1980,1983) estudiaron las debilidades de minimax aplicado a la aproximación de las evaluaciones. Ellos mostraron que bajo ciertos axiomas de independencia sobre las distribuciones de los valores de las hojas, minimaximizar puede producir valores en la raíz que son realmente *menos* fiables que el uso directo de la función de evaluación. El libro de Pearl, *Heuristics* (1984), explica parcialmente esta paradoja aparente y analiza muchos algoritmos de juegos. Baum y Smith (1997) proponen una sustitución a base de probabilidad para minimax, y muestra que ésto causa mejores opciones en ciertos juegos. Hay todavía poca teoría sobre los efectos de cortar la búsqueda en niveles diferentes y aplicar funciones de evaluación.

El algoritmo minimax esperado fue propuesto por Donald Michie (1966), aunque por supuesto sigue directamente los principios de evaluación de los árboles de juegos debido a von Neumann y Morgenstern. Bruce Ballard (1983) amplió la poda alfa-beta para cubrir árboles de nodos de posibilidad. El primer programa de *backgammon* fue BKG (Berliner, 1977,1980b); utilizó una función de evaluación compleja construida a mano y buscó sólo a profundidad 1. Fue el primer programa que derrotó a un campeón mundial humano en un juego clásico importante (Berliner, 1980a). Berliner reconoció que éste fue un partido de exhibición muy corto (no fue un partido del campeonato mundial) y que BKG tuvo mucha suerte con los dados. El trabajo de Gerry Tesauro, primero sobre NEUROGAMMON (Tesauro, 1989) y más tarde sobre TD-GAMMON (Tesauro, 1995), mostró que se pueden obtener muchos mejores resultados mediante el aprendizaje por refuerzo, que trataremos en el Capítulo 21.

Las damas, más que el ajedrez, fue el primer juego clásico jugado completamente por un computador. Christopher Strachey (1952) escribió el primer programa de funcionamiento para las damas. Schaeffer (1997) dio una muy legible, «con todas sus imperfecciones», cuenta del desarrollo de su programa de damas campeón del mundo Chinook.

Los primeros programas de Go fueron desarrollados algo más tarde que los de las damas y el ajedrez (Lefkovitz, 1960; Remus, 1962) y han progresado más despacio. Ryder (1971) usó una aproximación basada en la búsqueda pura con una variedad de métodos

de poda selectivos para vencer el enorme factor de ramificación. Zobrist (1970) usó las reglas condición-acción para sugerir movimientos plausibles cuando aparecieran los modelos conocidos. Reitman y Wilcox (1979) combinan reglas y búsqueda con efectos buenos, y los programas más modernos han seguido esta aproximación híbrida. Müller (2002) resume el estado del arte de la informatización de Go y proporciona una riqueza de referencias. Anshelevich (2000) utilizó las técnicas relacionadas para el juego Hex. *Computer Go Newsletter*, publicada por la Asociación de Go por computador, describe el desarrollo actual del juego.

Los trabajos sobre juegos de computador aparecen en multitud de sitios. La mal llamada conferencia *Heuristic Programming in Artificial Intelligence* hizo un informe sobre las Olimpiadas de Computador, que incluyen una amplia variedad de juegos. Hay también varias colecciones de trabajos importantes sobre la investigación en juegos (Levy, 1988a, 1988b; Marsland y Schaeffer, 1990). La Asociación Internacional de Ajedrez por Computador (ICCA), fundada en 1977, publica la *revista* trimestral *ICGA* (anteriormente la *revista ICCA*). Los trabajos importantes han sido publicados en la serie antológica *Advances in Computer Chess*, que comienza con Clarke (1977). El volumen 134 de la revista *Artificial Intelligence* (2002) contiene descripciones de programas para el ajedrez, Otelo, Hex, shogi, Go, *backgammon*, poker, Scrabble™ y otros juegos.

EJERCICIOS

6.1 En este problema se ejercitan los conceptos básicos de juegos, utilizando tic-tac-toe (tres en raya) como un ejemplo. Definimos X_n como el número de filas, columnas, o diagonales con exactamente *n Xs* y ningún O. Del mismo modo, O_n es el número de filas, columnas, o diagonales con solamente *n Os*. La función de utilidad asigna $+1$ a cualquier posición con $X_3 = 1$ y -1 a cualquier posición con $O_3 = 1$. Todas las otras posiciones terminales tienen utilidad 0. Para posiciones no terminales, usamos una función de evaluación lineal definida como $Eval(s) = 3\,X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$.

a) ¿Aproximadamente cuántos juegos posibles de tic-tac-toe hay?

b) Muestre el árbol de juegos entero hasta profundidad 2 (es decir, un *X* y un *O* sobre el tablero) comenzando con un tablero vacío, teniendo en cuenta las simetrías.

c) Señale sobre el árbol las evaluaciones de todas las posiciones a profundidad 2.

d) Usando el algoritmo minimax, marque sobre su árbol los valores hacia atrás para las posiciones de profundidades 1 y 0, y use esos valores para elegir el mejor movimiento de salida.

e) Marque los nodos a profundidad 2 que no serían evaluados si se aplicara la poda alfa-beta, asumiendo que los nodos están generados *en orden óptimo por la poda alfa-beta*.

6.2 Demuestre la afirmación siguiente: para cada árbol de juegos, la utilidad obtenida por MAX usando las decisiones minimax contra MIN subóptimo nunca será inferior que la utilidad obtenida jugando contra MIN óptimo. ¿Puede proponer un árbol de juegos en el cual MAX puede mejorar utilizando una estrategia subóptima contra MIN subóptimo?

6.3 Considere el juego de dos jugadores descrito en la Figura 6.14.

a) Dibuje el árbol de juegos completo, usando las convenciones siguientes:

- Escriba cada estado como (s_A, s_B) donde s_A y s_B denotan las posiciones simbólicas.
- Ponga cada estado terminal en una caja cuadrada y escriba su valor de juego en un círculo.
- Ponga los *estados bucle* (estados que ya aparecen sobre el camino a la raíz) en dobles cajas cuadradas. Ya que no está claro cómo adjudicar valores a estados bucle, anote cada uno con un «?» en un círculo.

b) Ahora marque cada nodo con su valor minimax hacia atrás (también en un círculo). Explique cómo maneja los valores «?» y por qué.

c) Explique por qué el algoritmo minimax estándar fallaría sobre este árbol de juegos y brevemente esboce cómo podría arreglarlo, usando su respuesta en (b). ¿Su algoritmo modificado proporciona las decisiones óptimas para todos los juegos con bucles?

d) Este juego de 4 cuadrados puede generalizarse a n cuadrados para cualquier $n > 2$. Demuestre que A gana si n es par y pierde si n es impar.

6.4 Implemente los generadores de movimiento y las funciones de evaluación para uno o varios de los juegos siguientes: Kalah, Otelo, damas y ajedrez. Construya un agente de juegos alfa-beta general que use su implementación. Compare el efecto de incrementar la profundidad de la búsqueda, mejorando el orden de movimientos, y la mejora de la función de evaluación. ¿Cuál es el factor de ramificación eficaz para el caso ideal de la ordenación perfecta de movimientos?

6.5 Desarrolle una demostración formal de la exactitud para la poda alfa-beta. Para hacer esto, considere la situación de la Figura 6.15. La pregunta es si hay que podar el nodo n_j, qué es un nodo max y un descendiente del nodo n_1. La idea básica es podarlo si y sólo si el valor minimax de n_1 puede demostrarse que es independiente del valor de n_j.

a) El valor de n_1 esta dado por

$$n_1 = \min (n_2, n_{21} \ldots, n_{2b_2})$$

Encuentre una expresión similar para n_2 y de ahí una expresión para n_1 en términos de n_j.

Figura 6.14 La posición de partida de un juego sencillo. El jugador A mueve primero. Los dos jugadores mueven por turno, y cada jugador debe mover su señal a un espacio vacio adyacente en *una u otra dirección*. Si el adversario ocupa un espacio adyacente, entonces un jugador puede saltar sobre el adversario al siguiente espacio vacio si existe. (Por ejemplo, si A está sobre 3 y B está sobre 2, entonces A puede mover hacia atrás a 1.) El juego se termina cuando un jugador alcanza el extremo opuesto del tablero. Si el jugador A alcanza el espacio 4 primero, el valor del juego a A es $+1$; si el jugador B alcanza el espacio 1 primero, entonces el valor del juego para A es -1.

b) Sea l_i el valor mínimo (o máximo) de los nodos a la *izquierda* del nodo n_i a profundidad i, cuyo valor minimax es ya conocido. Del mismo modo, sea r_i el valor mínimo (o máximo) de los nodos inexplorados de la derecha de n_i a profundidad i. Rescriba la expresión para n_1 en términos de los valores de r_i y l_i.

c) Ahora reformule la expresión para demostrar que para afectar a n_1, n_j no debe exceder de una cierta cota obtenida de los valores de l_i.

d) Repita el proceso para el caso donde n_j es un nodo min.

6.6 Implemente el algoritmo minimax esperado y el algoritmo *-alfa-beta, descrito por Ballard (1983), para podar árboles de juegos con nodos de posibilidad. Inténtelo sobre un juego como el *backgammon* y mida la eficacia de la poda *-alfa-beta.

6.7 Demuestre que con una transformación positiva lineal de valores de las hojas (es decir, transformando un valor x a $ax + b$ donde $a > 0$), la opción del movimiento permanece sin alterar en un árbol de juegos, aun cuando haya nodos posibilidad.

6.8 Considere el procedimiento siguiente para elegir movimientos en juegos con nodos de posibilidad:

- Genere algunas secuencias de lanzamientos de un dado (digamos, 50) a una profundidad conveniente (digamos, 8).
- Conocidos los lanzamientos del dado, el árbol de juegos se hace determinista. Para cada secuencia de lanzamientos del dado, resuelva el árbol de juegos determinista que ha resultado utilizando alfa-beta.
- Use los resultados para estimar el valor de cada movimiento y elegir el mejor.

¿Trabajará este procedimiento bien? ¿Por qué (no)?

6.9 Describa e implemente un entorno de juegos multijugador en tiempo real, donde el tiempo es parte del estado del ambiente y a los jugadores se le dan asignaciones de tiempo fijas.

Figura 6.15 Situación cuando consideramos si hay que podar el nodo n_j.

6.10 Describa o implemente las descripciones de los estados, generadores de movimiento, test terminal, función de utilidad y funciones de evaluación para uno o varios de los juegos siguientes: monopoly, scrabble, bridge (asumiendo un contrato dado), y póker (elija su variedad favorita).

6.11 Considere con cuidado la interacción de acontecimientos de posibilidad e información parcial en cada uno de los juegos del Ejercicio 6.10.

a) ¿Para cuáles es apropiado el minimax esperado estándar? Implemente el algoritmo y ejecútelo en su agente de juegos, con las modificaciones apropiadas al ambiente de juegos.
b) ¿Para cuál es apropiado el esquema descrito en el Ejercicio 6.8?
c) Discuta cómo podría tratar con el hecho que en algunos juegos, los jugadores no tienen el mismo conocimiento del estado actual.

6.12 El algoritmo minimax supone que los jugadores mueven por turnos, pero en juegos de cartas como *whist* y bridge, el ganador de la baza anterior juega primero sobre la siguiente baza.

a) Modifique el algoritmo para trabajar correctamente para estos juegos. Se supone que está disponible una función GANADOR(baza) que hace un informe sobre qué carta gana una baza.
b) Dibuje el árbol de juegos para el primer par de manos de la página 200.

6.13 El programa de damas Chinook hace uso de bases de datos de final del juego, que proporcionan valores exactos para cada posición con ocho o menos piezas. ¿Cómo podrían generarse tales bases de datos de manera eficiente?

6.14 Discuta cómo la aproximación estándar de juegos se aplicaría a juegos como tenis, billar y croquet, que ocurren en un espacio de estado físico continuo.

6.15 Describa cómo los algoritmos minimax y alfa-beta cambian en **juegos de suma no cero** de dos jugadores en los que cada jugador tiene su propia función utilidad. Podríamos suponer que cada jugador sabe la función de utilidad del otro. Si no hay restricciones sobre las dos utilidades terminales, ¿es posible podar algún nodo con alfa-beta?

6.16 Suponga que tiene un programa de ajedrez que puede evaluar un millón de nodos por segundo. Decida una representación de un estado del juego para almacenarlo en una tabla de transposiciones. ¿Sobre cuántas entradas puede poner en una tabla de 500MB de memoria? ¿Será suficiente tres minutos de búsqueda para un movimiento? ¿Cuántas consultas de la tabla puede hacer en el tiempo utilizado para hacer una evaluación? Ahora suponga que la tabla de transposiciones es más grande que la que puede caber en memoria. ¿Sobre cuántas evaluaciones podría hacer en el tiempo utilizado para realizar una búsqueda en disco con un disco estándar?

7 Agentes lógicos

Donde diseñaremos agentes que pueden construir representaciones del mundo, utilizar un proceso de inferencia para derivar nuevas representaciones del mundo, y emplear éstas para deducir qué hacer.

En este capítulo se introducen los agentes basados en conocimiento. Los conceptos que discutiremos (la *representación* del conocimiento y los procesos de *razonamiento* que permiten que éste evolucione) son centrales en todo el ámbito de la inteligencia artificial.

De algún modo, las personas conocen las cosas y realizan razonamientos. Tanto el conocimiento como el razonamiento son también importantes para los agentes artificiales, porque les permiten comportamientos con éxito que serían muy difíciles de alcanzar mediante otros mecanismos. Ya hemos visto cómo el conocimiento acerca de los efectos de las acciones permiten a los agentes que resuelven problemas actuar correctamente en entornos complejos. Un agente reflexivo sólo podría hallar un camino de Arad a Bucarest mediante la suerte del principiante. Sin embargo, el conocimiento de los agentes que resuelven problemas es muy específico e inflexible. Un programa de ajedrez puede calcular los movimientos permitidos de su rey, pero no puede saber de ninguna manera que una pieza no puede estar en dos casillas diferentes al mismo tiempo. Los agentes basados en conocimiento se pueden aprovechar del conocimiento expresado en formas muy genéricas, combinando y recombinando la información para adaptarse a diversos propósitos. A veces, este proceso puede apartarse bastante de las necesidades del momento (como cuando un matemático demuestra un teorema o un astrónomo calcula la esperanza de vida de la Tierra).

El conocimiento y el razonamiento juegan un papel importante cuando se trata con entornos parcialmente observables. Un agente basado en conocimiento puede combinar

el conocimiento general con las percepciones reales para inferir aspectos ocultos del estado del mundo, antes de seleccionar cualquier acción. Por ejemplo, un médico diagnostica a un paciente (es decir, infiere una enfermedad que no es directamente observable) antes de seleccionar un tratamiento. Parte del conocimiento que utiliza el médico está en forma de reglas que ha aprendido de los libros de texto y sus profesores, y parte en forma de patrones de asociación que el médico no es capaz de describir explícitamente. Si este conocimiento está en la cabeza del médico, es su conocimiento.

El entendimiento del lenguaje natural también necesita inferir estados ocultos, en concreto, la intención del que habla. Cuando escuchamos, «John vió el diamante a través de la ventana y lo codició», sabemos que «lo» se refiere al diamante y no a la ventana (quizá de forma inconsciente, razonamos con nuestro conocimiento acerca del papel relativo de las cosas). De forma similar, cuando escuchamos, «John lanzó el ladrillo a la ventana y se rompió», sabemos que «se» se refiere a la ventana. El razonamiento nos permite hacer frente a una variedad virtualmente infinita de manifestaciones utilizando un conjunto finito de conocimiento de sentido común. Los agentes que resuelven problemas presentan dificultades con este tipo de ambigüedad debido a que su representación de los problemas con contingencias es inherentemente exponencial.

Nuestra principal razón para estudiar los agentes basados en conocimiento es su flexibilidad. Ellos son capaces de aceptar tareas nuevas en forma de objetivos descritos explícitamente, pueden obtener rápidamente competencias informándose acerca del conocimiento del entorno o aprendiéndolo, y pueden adaptarse a los cambios del entorno actualizando el conocimiento relevante.

En la Sección 7.1 comenzamos con el diseño general del agente. En la Sección 7.2 se introduce un nuevo entorno muy sencillo, el mundo de *wumpus*, y se muestra la forma de actuar de un agente basado en conocimiento sin entrar en los detalles técnicos. Entonces, en la Sección 7.3, explicamos los principios generales de la **lógica**. La lógica será el instrumento principal para la representación del conocimiento en toda la Parte III de este libro. El conocimiento de los agentes lógicos siempre es *categórico* (cada proposición acerca del mundo es verdadera o falsa, si bien, el agente puede ser agnóstico acerca de algunas proposiciones).

La lógica presenta la ventaja pedagógica de ser un ejemplo sencillo de representación para los agentes basados en conocimiento, pero tiene serias limitaciones. En concreto, gran parte del razonamiento llevado a cabo por las personas y otros agentes en entornos parcialmente observables se basa en manejar conocimiento que es *incierto*. La lógica no puede representar bien esta incertidumbre, así que trataremos las probabilidades en la Parte V, que sí puede. Y en la Parte VI y la Parte VII trataremos otras representaciones, incluidas algunas basadas en matemáticas continuas como combinaciones de funciones Gaussianas, redes neuronales y otras representaciones.

En la Sección 7.4 de este capítulo se presenta una lógica muy sencilla denominada **lógica proposicional**. Aunque es mucho menos expresiva que la **lógica de primer orden** (Capítulo 8), la lógica proposicional nos permitirá ilustrar los conceptos fundamentales de la lógica. En las secciones 7.5 y 7.6 describiremos la tecnología, que está bastante desarrollada, para el razonamiento basado en lógica proposicional. Finalmente, en la sección 7.7 se combina el concepto de agente lógico con la tecnología de la lógica proposicional para la construcción de unos agentes muy sencillos en nuestro ejemplo

del mundo de *wumpus*. Se identifican ciertas deficiencias de la lógica proposicional, que nos permitirán el desarrollo de lógicas más potentes en los capítulos siguientes.

7.1 Agentes basados en conocimiento

BASE DE CONOCIMIENTO

SENTENCIA

LENGUAJE DE REPRESENTACIÓN DEL CONOCIMIENTO

El componente principal de un agente basado en conocimiento es su **base de conocimiento**, o BC. Informalmente, una base de conocimiento es un conjunto de **sentencias**. (Aquí «sentencia» se utiliza como un término técnico. Es parecido, pero no idéntico, a las sentencias en inglés u otros lenguajes naturales.) Cada sentencia se expresa en un lenguaje denominado **lenguaje de representación del conocimiento** y representa alguna aserción acerca del mundo.

INFERENCIA

AGENTES LÓGICOS

Debe haber un mecanismo para añadir sentencias nuevas a la base de conocimiento, y uno para preguntar qué se sabe en la base de conocimiento. Los nombres estándar para estas dos tareas son DECIR y PREGUNTAR, respectivamente. Ambas tareas requieren realizar **inferencia**, es decir, derivar nuevas sentencias de las antiguas. En los **agentes lógicos**, que son el tema principal de estudio de este capítulo, la inferencia debe cumplir con el requisito esencial de que cuando se PREGUNTA a la base de conocimiento, la respuesta debe seguirse de lo que se HA DICHO a la base de conocimiento previamente. Más adelante, en el capítulo, seremos más precisos en cuanto a la palabra «seguirse». Por ahora, tómate su significado en el sentido de que la inferencia no se inventaría cosas poco a poco.

CONOCIMIENTO DE ANTECEDENTES

La Figura 7.1 muestra el esquema general de un programa de un agente basado en conocimiento. Al igual que todos nuestros agentes, éste recibe una percepción como entrada y devuelve una acción. El agente mantiene una base de conocimiento, *BC*, que inicialmente contiene algún **conocimiento de antecedentes**. Cada vez que el programa del agente es invocado, realiza dos cosas. Primero, DICE a la base de conocimiento lo que ha percibido. Segundo, PREGUNTA a la base de conocimiento qué acción debe ejecutar. En este segundo proceso de responder a la pregunta, se debe realizar un razonamiento extensivo acerca del estado actual del mundo, de los efectos de las posibles acciones, etcétera. Una vez se ha escogido la acción, el agente graba su elección mediante un DECIR y ejecuta la acción. Este segundo DECIR es necesario para permitirle a la base de conocimiento saber que la acción hipotética realmente se ha ejecutado.

```
función AGENTE-BC(percepción) devuelve una acción
   variables estáticas: BC, una base de conocimiento
                        t, un contador, inicializado a 0, que indica el tiempo

   DECIR(BC, CONSTRUIR-SENTENCIA-DE-PERCEPCIÓN(percepción, t))
   acción ← PREGUNTAR(BC, PEDIR-Acción(t))
   DECIR(BC, CONSTRUIR-SENTENCIA-DE-ACCIÓN(acción, t))
   t ← t + 1
   devolver acción
```

Figura 7.1 Un agente basado en conocimiento genérico.

Los detalles del lenguaje de representación están ocultos en las dos funciones que implementan la interfaz entre los sensores, los accionadores, el núcleo de representación y el sistema de razonamiento. CONSTRUIR-SENTENCIA-DE-PERCEPCIÓN toma una percepción y un instante de tiempo y devuelve una sentencia afirmando lo que el agente ha percibido en ese instante de tiempo. PEDIR-ACCIÓN toma un instante de tiempo como entrada y devuelve una sentencia para preguntarle a la base de conocimiento qué acción se debe realizar en ese instante de tiempo. Los detalles de los mecanismos de inferencia están ocultos en DECIR y PREGUNTAR. En las próximas secciones del capítulo se mostrarán estos detalles.

El agente de la Figura 7.1 se parece bastante a los agentes con estado interno descritos en el Capítulo 2. Pero gracias a las definiciones de DECIR y PREGUNTAR, el agente basado en conocimiento no obtiene las acciones mediante un proceso arbitrario. Es compatible con una descripción al **nivel de conocimiento**, en el que sólo necesitamos especificar lo que el agente sabe y los objetivos que tiene para establecer su comportamiento. Por ejemplo, un taxi automatizado podría tener el objetivo de llevar un pasajero al condado de Marin, y podría saber que está en San Francisco y que el puente Golden Gate es el único enlace entre las dos localizaciones. Entonces podemos esperar que el agente cruce el puente Golden Gate *porque él sabe que hacerlo le permitirá alcanzar su objetivo*. Fíjate que este análisis es independiente de cómo el taxi trabaja al **nivel de implementación**. Al agente no le debe importar si el conocimiento geográfico está implementado mediante listas enlazadas o mapas de píxeles, o si su razonamiento se realiza mediante la manipulación de textos o símbolos almacenados en registros, o mediante la propagación de señales en una red de neuronas.

NIVEL DE CONOCIMIENTO

NIVEL DE IMPLEMENTACIÓN

Tal como comentamos en la introducción del capítulo, *uno puede construir un agente basado en conocimiento simplemente DICIÉNDOLE al agente lo que necesita saber*. El programa del agente, inicialmente, antes de que empiece a recibir percepciones, se construye mediante la adición, una a una, de las sentencias que representan el conocimiento del entorno que tiene el diseñador. El diseño del lenguaje de representación que permita, de forma más fácil, expresar este conocimiento mediante sentencias simplifica muchísimo el problema de la construcción del agente. Este enfoque en la construcción de sistemas se denomina **enfoque declarativo**. Por el contrario, el enfoque procedural codifica los comportamientos que se desean obtener directamente en código de programación; mediante la minimización del papel de la representación explícita y del razonamiento se pueden obtener sistemas mucho más eficientes. En la Sección 7.7 veremos agentes de ambos tipos. En los 70 y 80, defensores de los dos enfoques se enfrentaban en acalorados debates. Ahora sabemos que para que un agente tenga éxito su diseño debe combinar elementos declarativos y procedurales.

ENFOQUE DECLARATIVO

A parte de DECIRLE al agente lo que necesita saber, podemos proveer a un agente basado en conocimiento de los mecanismos que le permitan aprender por sí mismo. Estos mecanismos, que se verán en el Capítulo 18, crean un conocimiento general acerca del entorno con base en un conjunto de percepciones. Este conocimiento se puede incorporar a la base de conocimiento del agente y utilizar para su toma de decisiones. De esta manera, el agente puede ser totalmente autónomo.

Todas estas capacidades (representación, razonamiento y aprendizaje) se apoyan en la teoría y tecnología de la lógica, desarrolladas a lo largo de los siglos. Sin embargo,

antes de explicar dichas teoría y tecnología, crearemos un mundo sencillo que nos permitirá ilustrar estos mecanismos.

7.2 El mundo de *wumpus*

MUNDO DE *WUMPUS*

El **mundo de *wumpus*** es una cueva que está compuesta por habitaciones conectadas mediante pasillos. Escondido en algún lugar de la cueva está el *wumpus*, una bestia que se come a cualquiera que entre en su habitación. El *wumpus* puede ser derribado por la flecha de un agente, y éste sólo dispone de una. Algunas habitaciones contienen hoyos sin fondo que atrapan a aquel que deambula por dichas habitaciones (menos al *wumpus,* que es demasiado grande para caer en ellos). El único premio de vivir en este entorno es la posibilidad de encontrar una pila de oro. Aunque el mundo de *wumpus* pertenece más al ámbito de los juegos por computador, es un entorno perfecto para evaluar los agentes inteligentes. Michael Genesereth fue el primero que lo propuso.

En la Figura 7.2 se muestra un ejemplo del mundo de *wumpus*. La definición precisa del entorno de trabajo, tal como sugerimos en el Capítulo 2, mediante la descripción REAS, es:

- **Rendimiento:** $+1.000$ por recoger el oro, -1.000 por caer en un hoyo o ser comido por el *wumpus*, -1 por cada acción que se realice y -10 por lanzar la flecha.
- **Entorno:** una matriz de 4×4 habitaciones. El agente siempre comienza en la casilla etiquetada por [1, 1], y orientado a la derecha. Las posiciones del oro y del *wumpus* se escogen de forma aleatoria, mediante una distribución uniforme, a partir de todas las casillas menos la de salida del agente. Además, con probabilidad 0,2, cada casilla puede tener un hoyo.
- **Actuadores:** el agente se puede mover hacia delante, girar a la izquierda 90°, o a la derecha 90°. El agente puede fallecer de muerte miserable si entra en una casilla en la que hay un hoyo o en la que está el *wumpus* vivo. (No sucede nada malo, aunque huele bastante mal, si el agente entra en una casilla con un *wumpus* muerto.) Si hay un muro en frente y el agente intenta avanzar, no sucede nada. La acción *Agarrar* se puede utilizar para tomar un objeto de la misma casilla en donde se encuentre el agente. La acción *Disparar* se puede utilizar para lanzar una flecha en línea recta, en la misma dirección y sentido en que se encuentra situado el agente. La flecha avanza hasta que se choca contra un muro o alcanza al *wumpus* (y entonces lo mata). El agente sólo dispone de una flecha, así que, sólo tiene efecto el primer *Disparo*.
- **Sensores:** el agente dispone de cinco sensores, y cada uno le da una pequeña información acerca del entorno.
  - El agente percibirá un mal hedor si se encuentra en la misma casilla que el *wumpus* o en las directamente adyacentes a él (no en diagonal).
  - El agente recibirá una pequeña brisa en las casillas directamente adyacentes donde hay un hoyo.
  - El agente verá un resplandor en las casillas donde está el oro.

— Si el agente intenta atravesar un muro sentirá un golpe.
— Cuando el *wumpus* es aniquilado emite un desconsolado grito que se puede oír en toda la cueva.

Las percepciones que recibirá el agente se representan mediante una lista de cinco símbolos: por ejemplo, si el agente percibe un mal hedor o una pequeña brisa, pero no ve un resplandor, no siente un golpe, ni oye un grito, el agente recibe la lista [*Hedor*, *Brisa*, *Nada*, *Nada*, *Nada*].

En el Ejercicio 7.1 se pide definir el entorno del *wumpus* a partir de las diferentes dimensiones tratadas en el Capítulo 2. La principal dificultad para el agente es su ignorancia inicial acerca de la configuración del entorno; para superar esta ignorancia parece que se requiere el razonamiento lógico. En muchos casos del mundo de *wumpus*, para el agente es posible obtener el oro de forma segura. En algunos casos, el agente debe escoger entre volver a casa con las manos vacías o arriesgarse para encontrar el oro. Cerca del 21 por ciento de los casos son completamente injustos, ya que el oro se encuentra en un hoyo o rodeado de ellos.

Vamos a ver un agente basado en conocimiento en el mundo de *wumpus,* explorando el entorno que se muestra en la Figura 7.2. La base de conocimiento inicial del agente contiene las reglas del entorno, tal como hemos listado anteriormente; en concreto, el agente sabe que se encuentra en la casilla [1, 1] y que ésta es una casilla segura. Veremos cómo su conocimiento evoluciona a medida que recibe nuevas percepciones y las acciones se van ejecutando.

La primera percepción es [*Nada, Nada, Nada, Nada, Nada*], de la cual, el agente puede concluir que las casillas vecinas son seguras. La Figura 7.3(a) muestra el conocimiento del estado del agente en ese momento. En esta figura mostramos (algunas de) las sentencias de la base de conocimiento utilizando letras como la *B* (de brisa) y *OK* (de casilla segura, no hay hoyo ni está el *wumpus*) situadas en las casillas adecuadas. En cambio, la Figura 7.2 muestra el mundo tal como es.

Figura 7.2 Un mundo de wumpus típico. El agente está situado en la esquina inferior izquierda.

De los hechos, que no hay mal hedor ni brisa en la casilla [1, 1], el agente infiere que las casillas [1, 2] y [2, 1] están libres de peligro. Entonces las marca con *OK* para indicar esta conclusión. Un agente que sea cauto sólo se moverá hacia una casilla en la que él sabe que está *OK*. Supongamos que el agente decide moverse hacia delante a la casilla [2,1], alcanzando la situación de la Figura 7.3(b).

El agente detecta una brisa en la casilla [2, 1], por lo tanto, debe haber un hoyo en alguna casilla vecina. El hoyo no puede estar en la casilla [1, 1], teniendo en cuenta las reglas del juego, así que debe haber uno en la casilla [2, 2] o en la [3, 1], o en ambas. La etiqueta *¿P?* de la Figura 7.3(b), nos indica que puede haber un posible hoyo en estas casillas. En este momento, sólo se conoce una casilla que está *OK* y que no ha sido visitada aún. Así que el agente prudente girará para volver a la casilla [1, 1] y entonces se moverá a la [1, 2].

Figura 7.3 El primer paso dado por el agente en el mundo de *wumpus*. (a) La situación inicial, después de la percepción [*Nada, Nada, Nada, Nada, Nada*]. (b) Después del primer movimiento, con la percepción [*Nada, Brisa, Nada, Nada, Nada*].

La nueva percepción en la casilla [1, 2] es [*Hedor, Nada, Nada, Nada, Nada*], obteniendo el estado de conocimiento que se muestra en la Figura 7.4(a). El mal hedor en la [1, 2] significa que debe haber un *wumpus* muy cerca. Pero el *wumpus* no puede estar en la [1, 1], teniendo en cuenta las reglas del juego, y tampoco puede estar en [2, 2] (o el agente habría detectado un mal hedor cuando estaba en la [2, 1]). Entonces el agente puede inferir que el *wumpus* se encuentra en la casilla [1, 3], que se indica con la etiqueta *¡W!* Más aún, la ausencia de *Brisa* en la casilla [1, 2] implica que no hay un hoyo en la [2, 2]. Como ya habíamos inferido que debía haber un hoyo en la casilla [2, 2] o en la [3, 1], éste debe estar en la [3, 1]. Todo esto es un proceso de inferencia realmente costoso, ya que debe combinar el conocimiento adquirido en diferentes instantes de tiempo y en distintas situaciones, para resolver la falta de percepciones y poder realizar cualquier paso crucial. La inferencia pertenece a las habilidades de muchos animales, pero es típico del tipo de razonamiento que un agente lógico realiza.

Figura 7.4 Los dos últimos estados en el desarrollo del juego. (a) Después del tercer movimiento, con la percepción [*Hedor, Nada, Nada, Nada, Nada*]. (b) Después del quinto movimiento, con la percepción [*Hedor, Brisa, Resplandor, Nada, Nada*].

El agente ha demostrado en este momento que no hay ni un hoyo ni un *wumpus* en la casilla [2, 2], así que está *OK* para desplazarse a ella. No mostraremos el estado de conocimiento del agente en [2, 2]; asumimos que el agente gira y se desplaza a [2, 3], tal como se muestra en la Figura 7.4(b). En la casilla [2, 3] el agente detecta un resplandor, entonces el agente cogería el oro y acabaría el juego.

En cada caso en que el agente saca una conclusión a partir de la información que tiene disponible, se garantiza que dicha conclusión es correcta si la información disponible también lo es. Esta es una propiedad fundamental del razonamiento lógico. En lo que queda del capítulo vamos a describir cómo construir agentes lógicos que pueden representar la información necesaria para sacar conclusiones similares a las que hemos descrito en los párrafos anteriores.

7.3 Lógica

Esta sección presenta un repaso de todos los conceptos fundamentales de la representación y el razonamiento lógicos. Dejamos los detalles técnicos de cualquier clase concreta de lógica para la siguiente sección. En lugar de ello, utilizaremos ejemplos informales del mundo de *wumpus* y del ámbito familiar de la aritmética. Adoptamos este enfoque poco común, porque los conceptos de la lógica son bastante más generales y bellos de lo que se piensa a priori.

SINTAXIS

En la sección 7.1 dijimos que las bases de conocimiento se componen de sentencias. Estas sentencias se expresan de acuerdo a la **sintaxis** del lenguaje de representación, que especifica todas las sentencias que están bien formadas. El concepto de sintaxis está suficientemente claro en la aritmética: «$x + y = 4$» es una sentencia bien formada, mien-

tras que «$x2y+=$» no lo es. Por lo general, la sintaxis de los lenguajes lógicos (y la de los aritméticos, en cuanto al mismo tema) está diseñada para escribir libros y artículos. Hay literalmente docenas de diferentes sintaxis, algunas que utilizan muchas letras griegas y símbolos matemáticos complejos, otras basadas en diagramas con flechas y burbujas visualmente muy atractivas. Y sin embargo, en todos estos casos, las sentencias de la base de conocimiento del agente son configuraciones físicas reales (de las partes) del agente. El razonamiento implica generar y manipular estas configuraciones.

SEMÁNTICA

VALOR DE VERDAD

MUNDO POSIBLE

Una lógica también debe definir la **semántica** del lenguaje. Si lo relacionamos con el lenguaje hablado, la semántica trata el «significado» de las sentencias. En lógica, esta definición es bastante más precisa. La semántica del lenguaje define el **valor de verdad** de cada sentencia respecto a cada **mundo posible**. Por ejemplo, la semántica que se utiliza en la aritmética especifica que la sentencia «$x + y = 4$» es verdadera en un mundo en el que x sea 2 e y sea 2, pero falsa en uno en el que x sea 1 e y sea 1[1]. En las lógicas clásicas cada sentencia debe ser o bien verdadera o bien falsa en cada mundo posible, no puede ser lo uno y lo otro[2].

MODELO

Cuando necesitemos ser más precisos, utilizaremos el término **modelo** en lugar del de «mundo posible». (También utilizaremos la frase «m es un modelo de α» para indicar que la sentencia α es verdadera en el modelo m.) Siempre que podamos pensar en los mundos posibles como en (potencialmente) entornos reales en los que el agente pueda o no estar, los modelos son abstracciones matemáticas que simplemente nos permiten definir la verdad o falsedad de cada sentencia que sea relevante. Informalmente podemos pensar, por ejemplo, en que x e y son el número de hombres y mujeres que están sentados en una mesa jugando una partida de *bridge*, y que la sentencia $x + y = 4$ es verdadera cuando los que están jugando son cuatro en total; formalmente, los modelos posibles son justamente todas aquellas posibles asignaciones de números a las variables x e y. Cada una de estas asignaciones indica el valor de verdad de cualquier sentencia aritmética cuyas variables son x e y.

IMPLICACIÓN

Ahora que ya disponemos del concepto de valor de verdad, ya estamos preparados para hablar acerca del razonamiento lógico. Éste requiere de la relación de **implicación** lógica entre las sentencias (la idea de que una sentencia *se sigue lógicamente* de otra sentencia). Su notación matemática es

$$\alpha \models \beta$$

para significar que la sentencia α implica la sentencia β. La definición formal de implicación es esta: $\alpha \models \beta$ si y sólo si en cada modelo en el que α es verdadera, β también lo es. Otra forma de definirla es que si α es verdadera, β también lo debe ser. Informalmente, el valor de verdad de β «está contenido» en el valor de verdad de α. La relación de implicación nos es familiar en la aritmética; no nos disgusta la idea de que la sentencia $x + y = 4$ implica la sentencia $4 = x + y$. Es obvio que en cada modelo en

[1] El lector se habrá dado cuenta de la semejanza entre el concepto de valor de verdad de las sentencias y la satisfacción de restricciones del Capítulo 5. No es casualidad (los lenguajes de restricciones son en efecto lógicas y la resolución de restricciones un tipo de razonamiento lógico).

[2] La **lógica difusa**, que se verá en el Capítulo 14, nos permitirá tratar con grados de valores de verdad.

el que $x + y = 4$ (como lo es el modelo en el que x es 2 e y es 2) también lo es para $4 = x + y$. Pronto veremos que una base de conocimiento puede ser considerada como una afirmación, y a menudo hablaremos de que una base de conocimiento implica una sentencia.

Ahora podemos aplicar el mismo tipo de análisis que utilizamos en la sección anterior al mundo del *wumpus*. Si tomamos la situación de la Figura 7.3(b): el agente no ha detectado nada en la casilla [1, 1], y ha detectado una brisa en la [2, 1]. Estas percepciones, combinadas con el conocimiento del agente sobre las reglas que definen el funcionamiento del mundo de *wumpus* (la descripción REAS de la página 221), constituyen su BC. El agente está interesado (entre otras cosas) en si las casillas adyacentes [1, 2], [2, 2] y [3, 1] tienen hoyos sin fondo. Cada una de las tres casillas pueden o no tener un hoyo, por lo tanto (al menos en este ejemplo) hay $2^3 = 8$ modelos posibles. Tal como se muestran en la Figura 7.5[3].

La BC es falsa en los modelos que contradicen lo que el agente sabe (por ejemplo, la BC es falsa en cualquier modelo en el que la casilla [1, 2] tenga un hoyo), porque no ha detectado ninguna brisa en la casilla [1, 1]. De hecho, hay tres modelos en los que la BC es verdadera, los que se muestran como subconjunto de los modelos de la Figura 7.5. Ahora consideremos las dos conclusiones:

α_1 = «No hay un hoyo en la casilla [1, 2]».
α_2 = «No hay un hoyo en la casilla [2, 2]».

Figura 7.5 Modelos posibles para la presencia de hoyos en las casillas [1, 2], [2, 2] y [3, 1], dadas las observaciones de que no hay nada en la casilla [1, 1] y hay una brisa en la [2, 1]. (a) Modelos de la base de conocimiento y α_1 (no hay un hoyo en [1, 2]). (b) Modelos de la base de conocimiento y α_2 (no hay un hoyo en [2, 2]).

[3] En la Figura 7.5 los modelos se muestran como mundos parciales, porque en realidad tan sólo son asignaciones de *verdadero* y *falso* a sentencias como «hay un hoyo en la casilla [1, 2]», etc. Los modelos, desde el punto de vista matemático, no necesitan tener horribles wumpus etéreos ambulando en ellos.

Hemos rodeado (con línea discontinua) los modelos de α_1 y α_2 en las Figuras 7.5(a) y 7.5(b) respectivamente. Si observamos, podemos ver lo siguiente:

en cada modelo en el que la *BC* es verdadera, α_1 también lo es.

De aquí que $BC \models \alpha_1$: no hay un hoyo en la casilla [1, 2]. También podemos ver que

en algunos modelos en los que la *BC* es verdadera, α_2 es falsa.

De aquí que $BC \not\models \alpha_2$: el agente no puede concluir que no haya un hoyo en la casilla [2, 2]. (Ni tampoco puede concluir que lo haya.[4])

El ejemplo anterior no sólo nos muestra el concepto de implicación, sino, también cómo el concepto de implicación se puede aplicar para derivar conclusiones, es decir, llevar a cabo la **inferencia lógica**. El algoritmo de inferencia que se muestra en la Figura 7.5 se denomina **comprobación de modelos** porque enumera todos los modelos posibles y comprueba si α es verdadera en todos los modelos en los que la *BC* es verdadera.

INFERENCIA LÓGICA

COMPROBACIÓN DE MODELOS

Para entender la implicación y la inferencia nos puede ayudar pensar en el conjunto de todas las consecuencias de la *BC* como en un pajar, y en α como en una aguja. La implicación es como la aguja que se encuentra en el pajar, y la inferencia consiste en encontrarla. Esta distinción se expresa mediante una notación formal: si el algoritmo de inferencia *i* puede derivar α de la *BC*, entonces escribimos

$$BC \vdash_i \alpha,$$

que se pronuncia como «α se deriva de la *BC* mediante *i*» o «*i* deriva α de la *BC*».

Se dice que un algoritmo de inferencia que deriva sólo sentencias implicadas es **sólido** o que **mantiene la verdad**. La solidez es una propiedad muy deseable. Un procedimiento de inferencia no sólido tan sólo se inventaría cosas poco a poco (anunciaría el descubrimiento de agujas que no existirían). Se puede observar fácilmente que la comprobación de modelos, cuando es aplicable[5], es un procedimiento sólido.

SÓLIDO

MANTENIMIENTO DE LA VERDAD

COMPLETITUD

También es muy deseable la propiedad de **completitud**: un algoritmo de inferencia es completo si puede derivar cualquier sentencia que está implicada. En los pajares reales, que son de tamaño finito, parece obvio que un examen sistemático siempre permite decidir si hay una aguja en el pajar. Sin embargo, en muchas bases de conocimiento, el pajar de las consecuencias es infinito, y la completitud pasa a ser una problemática importante[6]. Por suerte, hay procedimientos de inferencia completos para las lógicas que son suficientemente expresivas para manejar muchas bases de conocimiento.

---

[4] El agente podría calcular la *probabilidad* de que haya un hoyo en la casilla [2, 2]; en el Capítulo 13 lo veremos.

[5] La comprobación de modelos trabaja bien cuando el espacio de los modelos es finito (por ejemplo, en un mundo del *wumpus* de tamaño de casillas fijo). Por el otro lado, en la aritmética, el espacio de modelos es infinito: aun limitándonos a los enteros, hay infinitos pares de valores para x e y en la sentencia $x + y = 4$.

[6] Compárelo con el caso de la búsqueda en espacios infinitos de estados del Capítulo 3, en donde la búsqueda del primero en profundidad no es completa.

Hemos descrito un proceso de razonamiento en el que se garantiza que las conclusiones sean verdaderas en cualquier mundo en el que las premisas lo sean; en concreto, *si una BC es verdadera en el mundo real, entonces cualquier sentencia α que se derive de la BC mediante un procedimiento de inferencia sólido también será verdadera en el mundo real*. Así, mientras que un proceso de inferencia opera con la «sintaxis» (las configuraciones físicas internas, tales como los bits en los registros o los patrones de impulsos eléctricos en el cerebro) el proceso se corresponde con la relación del mundo real según la cual algún aspecto del mundo real es cierto[7] en virtud de que otros aspectos del mundo real lo son. En la Figura 7.6 se ilustra esta correspondencia entre el mundo y la representación.

DENOTACIÓN

El último asunto que debe ser tratado mediante una computación basada en agentes lógicos es el de la **denotación** (la conexión, si la hay, entre los procesos de razonamiento lógico y el entorno real en el que se encuentra el agente). En concreto, *¿cómo sabemos que la BC es verdadera en el mundo real?* (Después de todo, la *BC* sólo es «sintaxis» dentro de la cabeza del agente.) Ésta es una cuestión filosófica acerca de la cual se han escrito muchos, muchísimos libros. (Ver Capítulo 26.) Una respuesta sencilla es que los sensores del agente crean la conexión. Por ejemplo, nuestro agente del mundo de *wumpus* dispone de un sensor de olores. El programa del agente crea una sentencia adecuada siempre que hay un olor. Entonces, siempre que esa sentencia esté en la base de conocimiento será verdadera en el mundo real. Así, el significado y el valor de verdad de las sentencias de las percepciones se definen mediante el proceso de los sensores y el de la construcción de las sentencias, activada por el proceso previo. ¿Qué sucede con el resto del conocimiento del agente, tal como sus creencias acerca de que el *wumpus* causa mal hedor en las casillas adyacentes? Ésta no es una representación directa de una simple percepción, pero sí es una regla general (derivada, quizá, de la experiencia de las percepciones aunque no idéntica a una afirmación de dicha experiencia). Las reglas generales como ésta se generan mediante un proceso de construcción de sentencias denominado **aprendizaje**, que es el tema que trataremos en la Parte VI. El aprendizaje es falible. Puede darse el caso en el que el *wumpus* cause mal hedor *excepto el 29 de febrero en años bisiestos*, que

Figura 7.6 Las sentencias son configuraciones físicas del agente, y el razonamiento es el proceso de construcción de nuevas configuraciones físicas a partir de las antiguas. El razonamiento lógico debería asegurar que las nuevas configuraciones representen aspectos del mundo que realmente se siguen de los aspectos que las antiguas configuraciones representan.

[7] Tal como escribió Wittgenstein (1922) en su famoso *Tractatus*: «El mundo es cada cosa que es cierta».

es cuando toma su baño. Así, la *BC* no sería verdadera en el mundo real, sin embargo, mediante procedimientos de aprendizaje buenos no hace falta ser tan pesimistas.

7.4 Lógica proposicional: una lógica muy sencilla

LÓGICA PROPOSICIONAL

Ahora vamos a presentar una lógica muy sencilla llamada **lógica proposicional**[8]. Vamos a cubrir tanto la sintaxis como la semántica (la manera como se define el valor de verdad de las sentencias) de la lógica proposicional. Luego trataremos la **implicación** (la relación entre una sentencia y la que se sigue de ésta) y veremos cómo todo ello nos lleva a un algoritmo de inferencia lógica muy sencillo. Todo ello tratado, por supuesto, en el mundo de *wumpus*.

Sintaxis

SENTENCIAS ATÓMICAS

SÍMBOLO PROPOSICIONAL

La **sintaxis** de la lógica proposicional nos define las sentencias que se pueden construir. Las **sentencias atómicas** (es decir, los elementos sintácticos indivisibles) se componen de un único **símbolo proposicional**. Cada uno de estos símbolos representa una proposición que puede ser verdadera o falsa. Utilizaremos letras mayúsculas para estos símbolos: P, Q, R, y siguientes. Los nombres de los símbolos suelen ser arbitrarios pero a menudo se escogen de manera que tengan algún sentido mnemotécnico para el lector. Por ejemplo, podríamos utilizar $W_{1,3}$ para representar que el *wumpus* se encuentra en la casilla [1, 3]. (Recuerde que los símbolos como $W_{1,3}$ son *atómicos*, esto es, W, 1, y 3 no son partes significantes del símbolo.) Hay dos símbolos proposicionales con significado fijado: *Verdadero*, que es la proposición que siempre es verdadera; y *Falso*, que es la proposición que siempre es falsa.

SENTENCIAS COMPLEJAS

CONECTIVAS LÓGICAS

NEGACIÓN

LITERAL

CONJUNCIÓN

DISYUNCIÓN

IMPLICACIÓN

PREMISA

CONCLUSIÓN

Las **sentencias complejas** se construyen a partir de sentencias más simples mediante el uso de las **conectivas lógicas**, que son las siguientes cinco:

- $\neg$ (no). Una sentencia como $\neg W_{1,3}$ se denomina **negación** de $W_{1,3}$. Un **literal** puede ser una sentencia atómica (un **literal positivo**) o una sentencia atómica negada (un **literal negativo**).
- $\wedge$ (y). Una sentencia que tenga como conectiva principal $\wedge$, como es $W_{1,3} \wedge H_{3,1}$, se denomina **conjunción**; sus componentes son los **conjuntores**.
- $\vee$ (o). Una sentencia que utiliza la conectiva $\vee$, como es $(W_{1,3} \wedge H_{3,1}) \vee W_{2,2}$, es una **disyunción** de los **disyuntores** $(W_{1,3} \wedge H_{3,1})$ y $W_{2,2}$. (Históricamente, la conectiva $\vee$ proviene de «vel» en Latín, que significa «o». Para mucha gente, es más fácil recordarla como la conjunción al revés.)
- $\Rightarrow$ (implica). Una sentencia como $(W_{1,3} \wedge H_{3,1}) \Rightarrow \neg W_{2,2}$ se denomina **implicación** (o condicional). Su **premisa** o **antecedente** es $(W_{1,3} \wedge H_{3,1})$, y su **conclusión** o **consecuente** es $\neg W_{2,2}$. Las implicaciones también se conocen como **reglas** o afir-

[8] A la lógica proposicional también se le denomina **Lógica Booleana**, por el matemático George Boole (1815-1864).

maciones **si-entonces**. Algunas veces, en otros libros, el símbolo de la implicación se representa mediante $\supset$ o $\rightarrow$.

BICONDICIONAL

$\Leftrightarrow$ (si y sólo si). La sentencia $W_{1,3} \Leftrightarrow \neg W_{2,2}$ es una **bicondicional**.

En la Figura 7.7 se muestra una gramática formal de la lógica proposicional; mira la página 984 si no estás familiarizado con la notación BNF.

Sentencia	$\rightarrow$	*Sentencia Atómica* | *Sentencia Compleja*
Sentencia Atómica	$\rightarrow$	**Verdadero** | **Falso** | *Símbolo Proposicional*
Símbolo Proposicional	$\rightarrow$	**P** | **Q** | **R** | **...**
Sentencia Compleja	$\rightarrow$	$\neg$ *Sentencia*
	|	(*Sentencia* $\wedge$ *Sentencia*)
	|	(*Sentencia* $\vee$ *Sentencia*)
	|	(*Sentencia* $\Rightarrow$ *Sentencia*)
	|	(*Sentencia* $\Leftrightarrow$ *Sentencia*)

Figura 7.7 Una gramática BNF (Backus-Naur Form) de sentencias en lógica proposicional.

Fíjese en que la gramática es muy estricta respecto al uso de los paréntesis: cada sentencia construida a partir de conectivas binarias debe estar encerrada en paréntesis. Esto asegura que la gramática no sea ambigua. También significa que tenemos que escribir, por ejemplo, $((A \wedge B) \Rightarrow C)$ en vez de $A \wedge B \Rightarrow C$. Para mejorar la legibilidad, a menudo omitiremos paréntesis, apoyándonos en su lugar en un orden de precedencia de las conectivas. Es una precedencia similar a la utilizada en la aritmética (por ejemplo, $ab + c$ se lee $((ab) + c)$ porque la multiplicación tiene mayor precedencia que la suma). El orden de precedencia en la lógica proposicional (de mayor a menor) es: $\neg$, $\wedge$, $\vee$, $\Rightarrow$ y $\Leftrightarrow$. Así, la sentencia

$$\neg P \vee Q \wedge R \Rightarrow S$$

es equivalente a la sentencia

$$((\neg P) \vee (Q \wedge R)) \Rightarrow S$$

La precedencia entre las conectivas no resuelve la ambigüedad en sentencias como $A \wedge B \wedge C$, que se podría leer como $((A \wedge B) \wedge C)$ o como $(A \wedge (B \wedge C))$. Como estas dos lecturas significan lo mismo según la semántica que mostraremos en la siguiente sección, se permiten este tipo de sentencias. También se permiten sentencias como $A \vee B \vee C$ o $A \Leftrightarrow B \Leftrightarrow C$. Sin embargo, las sentencias como $A \Rightarrow B \Rightarrow C$ no se permiten, ya que su lectura en una dirección y su opuesta tienen significados muy diferentes; en este caso insistimos en la utilización de los paréntesis. Por último, a veces utilizaremos corchetes, en vez de paréntesis, para conseguir una lectura de la sentencia más clara.

Semántica

Una vez especificada la sintaxis de la lógica proposicional, vamos a definir su semántica. La semántica define las reglas para determinar el valor de verdad de una sentencia respecto a un modelo en concreto. En la lógica proposicional un modelo define el va-

lor de verdad (*verdadero* o *falso)*. Por ejemplo, si las sentencias de la base de conocimiento utilizan los símbolos proposicionales $H_{1,2}$, $H_{2,2}$, y $H_{3,1}$, entonces un modelo posible sería

$$m_1 = \{H_{1,2} = falso, H_{2,2} = \text{falso}, H_{3,1} = verdadero\}$$

Con tres símbolos proposicionales hay $2^3 = 8$ modelos posibles, exactamente los que aparecen en la Figura 7.5. Sin embargo, fíjese en que gracias a que hemos concretado la sintaxis, los modelos se convierten en objetos puramente matemáticos sin tener necesariamente una conexión al mundo de *wumpus*. $H_{1,2}$ es sólo un símbolo, podría denotar tanto «hay un hoyo en la casilla [1, 2]», como «estaré en París hoy y mañana».

La semántica en lógica proposicional debe especificar cómo obtener el valor de verdad de *cualquier* sentencia, dado un modelo. Este proceso se realiza de forma recursiva. Todas las sentencias se construyen a partir de las sentencias atómicas y las cinco conectivas lógicas; entonces necesitamos establecer cómo definir el valor de verdad de las sentencias atómicas y cómo calcular el valor de verdad de las sentencias construidas con las cinco conectivas lógicas. Para las sentencias atómicas es sencillo:

- *Verdadero* es verdadero en todos los modelos y *Falso* es falso en todos los modelos.
- El valor de verdad de cada símbolo proposicional se debe especificar directamente para cada modelo. Por ejemplo, en el modelo anterior m_1, $H_{1,2}$ es falso.

Para las sentencias complejas, tenemos reglas como la siguiente

- Para toda sentencia s y todo modelo m, la sentencia $\neg s$ es verdadera en m si y sólo si s es falsa en m.

Este tipo de reglas reducen el cálculo del valor de verdad de una sentencia compleja al valor de verdad de las sentencias más simples. Las reglas para las conectivas se pueden resumir en una **tabla de verdad** que especifica el valor de verdad de cada sentencia compleja según la posible asignación de valores de verdad realizada a sus componentes. En la Figura 7.8 se muestra la tabla de verdad de las cinco conectivas lógicas. Utilizando estas tablas de verdad, se puede obtener el valor de verdad de cualquier sentencia s según un modelo m mediante un proceso de evaluación recursiva muy sencillo. Por ejemplo, la sentencia $\neg H_{1,2} \wedge (H_{2,2} \vee H_{3,1})$ evaluada según m_1, da *verda-*

TABLA DE VERDAD

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
falso	*falso*	*verdadero*	*falso*	*falso*	*verdadero*	*verdadero*
falso	*verdadero*	*verdadero*	*falso*	*verdadero*	*verdadero*	*falso*
verdadero	*falso*	*falso*	*falso*	*verdadero*	*falso*	*falso*
verdadero	*verdadero*	*falso*	*verdadero*	*verdadero*	*verdadero*	*verdadero*

Figura 7.8 Tablas de verdad para las cinco conectivas lógicas. Para utilizar la tabla, por ejemplo, para calcular el valor de $P \vee Q$, cuando P es verdadero y Q falso, primero mire a la izquierda en donde P es *verdadera* y Q es *falsa* (la tercera fila). Entonces mire en esa fila justo en la columna de $P \vee Q$ para ver el resultado: *verdadero*. Otra forma de verlo es pensar en cada fila como en un modelo, y que sus entradas en cada fila dicen para cada columna si la sentencia es verdadera en ese modelo.

dero $\wedge$ (*falso* $\vee$ *verdadero*) = *verdadero* $\wedge$ *verdadero* = *verdadero*. El Ejercicio 7.3 pide que escriba el algoritmo ¿V-VERDAD-LP?(*s, m*) que debe obtener el valor de verdad de una sentencia *s* en lógica proposicional según el modelo *m*.

Ya hemos comentado que una base de conocimiento está compuesta por sentencias. Ahora podemos observar que esa base de conocimiento lógica es una conjunción de dichas sentencias. Es decir, si comenzamos con una *BC* vacía y ejecutamos DECIR(BC, S_1)… DECIR(BC, S_n) entonces tenemos $BC = S_1 \wedge \ldots \wedge S_n$. Esto significa que podemos manejar bases de conocimiento y sentencias de manera intercambiable.

Los valores de verdad de «y», «o» y «no» concuerdan con nuestra intuición, cuando los utilizamos en lenguaje natural. El principal punto de confusión puede presentarse cuando $P \vee Q$ es verdadero porque *P* lo es, *Q* lo es, o *ambos* lo son. Hay una conectiva diferente denominada «o exclusiva» («xor» para abreviar) que es falsa cuando los dos disyuntores son verdaderos[9]. No hay consenso respecto al símbolo que representa la o exclusiva, siendo las dos alternativas $\dot{\vee}$ y $\oplus$.

El valor de verdad de la conectiva $\Rightarrow$ puede parecer incomprensible al principio, ya que no encaja en nuestra comprensión intuitiva acerca de «*P* implica *Q*» o de «si *P* entonces *Q*». Para una cosa, la lógica proposicional no requiere de una relación de causalidad o relevancia entre *P* y *Q*. La sentencia «que 5 sea impar implica que Tokio es la capital de Japón» es una sentencia verdadera en lógica proposicional (bajo una interpretación normal), aunque pensándolo es, decididamente, una frase muy rara. Otro punto de confusión es que cualquier implicación es verdadera siempre que su antecedente sea falso. Por ejemplo, «que 5 sea par implica que Sam es astuto» es verdadera, independientemente de que Sam sea o no astuto. Parece algo estrafalario, pero tiene sentido si piensa acerca de «$P \Rightarrow Q$» como si dijera, «si *P* es verdadero, entonces estoy afirmando que *Q* es verdadero. De otro modo, no estoy haciendo ninguna afirmación.» La única manera de hacer esta sentencia *falsa* es haciendo que *P* sea cierta y *Q* falsa.

La tabla de verdad de la bicondicional $P \Leftrightarrow Q$ muestra que la sentencia es verdadera siempre que $P \Rightarrow Q$ *y* $Q \Rightarrow P$ lo son. En lenguaje natural a menudo se escribe como «*P* si y sólo si *Q*» o «*P* si *Q*». Las reglas del mundo de *wumpus* se describen mejor utilizando la conectiva $\Leftrightarrow$. Por ejemplo, una casilla tiene corriente de aire *si* alguna casilla vecina tiene un hoyo, y una casilla tiene corriente de aire *sólo si* una casilla vecina tiene un hoyo. De esta manera necesitamos bicondicionales como

$$B_{1,1} \Leftrightarrow (H_{1,2} \vee H_{2,1}),$$

en donde $B_{1,1}$ significa que hay una brisa en la casilla [1,1]. Fíjese en que la implicación

$$B_{1,1} \Rightarrow (H_{1,2} \vee H_{2,1})$$

es verdadera, aunque incompleta, en el mundo de *wumpus*. Esta implicación no descarta modelos en los que $B_{1,1}$ sea falso y $H_{1,2}$ sea verdadero, hecho que violaría las reglas del mundo de *wumpus*. Otra forma de observar esta incompletitud es que la implicación necesita la presencia de hoyos si hay una corriente de aire, mientras que la bicondicional además necesita la ausencia de hoyos si no hay ninguna corriente de aire.

[9] En latín está la palabra específica *aut* para la o exclusiva.

Una base de conocimiento sencilla

Ahora que ya hemos definido la semántica de la lógica proposicional, podemos construir una base de conocimiento para el mundo de *wumpus*. Para simplificar, sólo trataremos con hechos y reglas acerca de hoyos; dejamos el tratamiento del *wumpus* como ejercicio. Vamos a proporcionar el conocimiento suficiente para llevar a cabo la inferencia que se trató en la Sección 7.3.

Primero de todo, necesitamos escoger nuestro vocabulario de símbolos proposicionales. Para cada *i, j*:

- Hacemos que $H_{i,j}$ sea verdadero si hay un hoyo en la casilla [*i, j*].
- Hacemos que $B_{i,j}$ sea verdadero si hay una corriente de aire (una brisa) en la casilla [i, j].

La base de conocimiento contiene, cada una etiquetada con un identificador, las siguientes sentencias:

- No hay ningún hoyo en la casilla [1, 1].

$$R_1: \quad \neg H_{1,1}$$

- En una casilla se siente una brisa si y sólo si hay un hoyo en una casilla vecina. Esta regla se ha de especificar para cada casilla; por ahora, tan sólo incluimos las casillas que son relevantes:

$$R_2: \quad B_{1,1} \quad \Leftrightarrow \quad (H_{1,2} \vee H_{2,1})$$
$$R_3: \quad B_{2,1} \quad \Leftrightarrow \quad (H_{1,1} \vee H_{2,2} \vee H_{3,1})$$

- Las sentencias anteriores son verdaderas en todos los mundos de *wumpus*. Ahora incluimos las percepciones de brisa para las dos primeras casillas visitadas en el mundo concreto en donde se encuentra el agente, llegando a la situación que se muestra en la Figura 7.3(b).

$$R_4: \quad \neg B_{1,1}$$
$$R_5: \quad B_{2,1}$$

Entonces, la base de conocimiento está compuesta por las sentencias R_1 hasta R_5. La *BC* también se puede representar mediante una única sentencia (la conjunción $R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$) porque dicha sentencia aserta que todas las sentencias son verdaderas.

Inferencia

Recordemos que el objetivo de la inferencia lógica es decidir si $BC \models \alpha$ para alguna sentencia α. Por ejemplo, si se deduce $H_{2,2}$. Nuestro primer algoritmo para la inferencia será una implementación directa del concepto de implicación: enumerar los modelos, y averiguar si α es verdadera en cada modelo en el que la *BC* es verdadera. En la lógica proposicional los modelos son asignaciones de los valores *verdadero* y *falso* sobre cada símbolo proposicional. Volviendo a nuestro ejemplo del mundo de *wumpus*, los símbolos proposicionales relevantes son $B_{1,1}$, $B_{2,1}$, $H_{1,1}$, $H_{1,2}$, $H_{2,1}$, $H_{2,2}$ y $H_{3,1}$. Con es-

tos siete símbolos, tenemos $2^7 = 128$ modelos posibles; y en tres de estos modelos, la *BC* es verdadera (Figura 7.9). En esos tres modelos $\neg H_{1,2}$ es verdadera, por lo tanto, no hay un hoyo en la casilla [1, 2]. Por el otro lado, $H_{2,2}$ es verdadera en dos de esos tres modelos y falsa en el tercero, entonces todavía no podemos decir si hay un hoyo en la casilla [2, 2].

La Figura 7.9 reproduce más detalladamente el razonamiento que se mostraba en la Figura 7.5. En la Figura 7.10 se muestra un algoritmo general para averiguar la implicación en lógica proposicional. De forma similar al algoritmo de BÚSQUEDA-CON-BACKTRACKING de la página 86, ¿IMPLICACIÓN-EN-TV? Realiza una enumeración recursiva de un espacio finito de asignaciones a variables. El algoritmo es **sólido** porque implemen-

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	BC
falso	*falso*	*falso*	*falso*	*falso*	*falso*	*falso*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*falso*	*falso*
falso	*falso*	*falso*	*falso*	*falso*	*falso*	*verdadero*	*verdadero*	*verdadero*	*falso*	*verdadero*	*falso*	*falso*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
falso	*verdadero*	*falso*	*falso*	*falso*	*falso*	*falso*	*verdadero*	*verdadero*	*falso*	*verdadero*	*verdadero*	*falso*
falso	*verdadero*	*falso*	*falso*	*falso*	*falso*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*
falso	*verdadero*	*falso*	*falso*	*falso*	*verdadero*	*falso*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*
falso	*verdadero*	*falso*	*falso*	*falso*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*
falso	*verdadero*	*falso*	*falso*	*verdadero*	*falso*	*falso*	*verdadero*	*falso*	*falso*	*verdadero*	*verdadero*	*falso*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
verdadero	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*verdadero*	*falso*	*verdadero*	*verdadero*	*falso*	*verdadero*	*falso*

Figura 7.9 Una tabla de verdad construida para la base de conocimiento del ejemplo. La *BC* es verdadera si R_1 hasta R_5 son verdaderas, cosa que sucede en tres de las 128 filas. En estas tres filas, $H_{1,2}$ es falsa, así que no hay ningún hoyo en la casilla [1, 2]. Por otro lado, puede haber (o no) un hoyo en la casilla [2, 2].

```
función ¿IMPLICACIÓN-EN-TV?(BC, α) devuelve verdadero o falso
   entradas: BC, la base de conocimiento, una sentencia en lógica proposicional
             α, la sentencia implicada, una sentencia en lógica proposicional

   símbolos ← una lista de símbolos proposicionales de la BC y α
   devuelve COMPROBAR-TV(BC, α, símbolos, [ ])
---------------------------------------------------------------------------
función COMPROBAR-TV(BC, α, símbolos, modelo) devuelve verdadero o falso
   si ¿VACÍA?(símbolos) entonces
      si ¿VERDADERO-LP?(BC, modelo) entonces devuelve ¿VERDADERO-LP?(α, modelo)
      sino devuelve verdadero
   sino hacer
      P ← PRIMERO(símbolos); resto ← RESTO(símbolos)
      devuelve CHEQUEAR-TV(BC, α, resto, EXTENDER(P, verdadero, modelo)) y
               COMPROBAR-TV(BC, α, resto, EXTENDER(P, falso, modelo))
```

Figura 7.10 Un algoritmo de enumeración de una tabla de verdad para averiguar la implicación proposicional. TV viene de tabla de verdad. ¿VERDADERO-LP? Devuelve verdadero si una sentencia es verdadera en un modelo. La variable *modelo* representa un modelo parcial (una asignación realizada a un subconjunto de las variables). La llamada a la función EXTENDER(*P*, *verdadero*, *modelo*) devuelve un modelo parcial nuevo en el que *P* tiene el valor de verdad *verdadero*.

ta de forma directa la definición de implicación, y es **completo** porque trabaja para cualquier *BC* y sentencia α, y siempre finaliza (sólo hay un conjunto finito de modelos a ser examinados).

Por supuesto, que «conjunto finito» no siempre es lo mismo que «pequeño». Si la *BC* y α contienen en total n símbolos, entonces tenemos 2^n modelos posibles. Así, la complejidad temporal del algoritmo es $O(2^n)$. (La complejidad espacial sólo es $O(n)$ porque la enumeración es en primero en profundidad.) Más adelante, en este capítulo, veremos algoritmos que en la práctica son mucho más eficientes. Desafortunadamente, *cada algoritmo de inferencia que se conoce en lógica proposicional tiene un caso peor, cuya complejidad es exponencial respecto al tamaño de la entrada*. No esperamos mejorarlo, ya que demostrar la implicación en lógica proposicional es un problema co-NP-completo. (*Véase* Apéndice A.)

Equivalencia, validez y *satisfacibilidad*

Antes de que nos sumerjamos en los detalles de los algoritmos de inferencia lógica necesitaremos algunos conceptos adicionales relacionados con la implicación. Al igual que la implicación, estos conceptos se aplican a todos los tipos de lógica, sin embargo, se entienden más fácilmente para una en concreto, como es el caso de la lógica proposicional.

EQUIVALENCIA LÓGICA

El primer concepto es la **equivalencia lógica**: dos sentencias α y β son equivalentes lógicamente si tienen los mismos valores de verdad en el mismo conjunto de modelos. Este concepto lo representamos con $\alpha \Leftrightarrow \beta$. Por ejemplo, podemos observar fácilmente (mediante una tabla de verdad) que $P \wedge Q$ y $Q \wedge P$ son equivalentes lógicamente. En la Figura 7.11 se muestran otras equivalencias. Éstas juegan el mismo papel en la lógica que las igualdades en las matemáticas. Una definición alternativa de equivalencia es la siguiente: para dos sentencias α y β cualesquiera,

$$\alpha \equiv \beta \quad \text{si y sólo si} \quad \alpha \models \beta \text{ y } \beta \models \alpha$$

(Recuerde que $\models$ significa implicación.)

$$\begin{array}{rcll}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) & \text{Conmutatividad de } \wedge \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) & \text{Conmutatividad de } \vee \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) & \text{Asociatividad de } \wedge \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) & \text{Asociatividad de } \vee \\
\neg(\neg\alpha) & \equiv & \alpha & \text{Eliminación de la doble negación} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{Contraposición} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) & \text{Eliminación de la implicación} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) & \text{Eliminación de la bicondicional} \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) & \text{Ley de Morgan} \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) & \text{Ley de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & \text{Distribución de } \wedge \text{ respecto a } \vee \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) & \text{Distribución de } \vee \text{ respecto a } \wedge
\end{array}$$

Figura 7.11 Equivalencias lógicas. Los símbolos α, β y γ se pueden sustituir por cualquier sentencia en lógica proposicional.

VALIDEZ

TAUTOLOGÍA

El segundo concepto que necesitaremos es el de **validez**. Una sentencia es válida si es verdadera en *todos* los modelos. Por ejemplo, la sentencia $P \vee \neg P$ es una sentencia válida. Las sentencias válidas también se conocen como **tautologías**, son *necesariamente* verdaderas y por lo tanto vacías de significado. Como la sentencia *Verdadero* es verdadera en todos los modelos, toda sentencia válida es lógicamente equivalente a *Verdadero*.

TEOREMA DE LA DEDUCCIÓN

¿Qué utilidad tienen las sentencias válidas? De nuestra definición de implicación podemos derivar el **teorema de la deducción**, que ya se conocía por los Griegos antiguos:

Para cualquier sentencia α y β, $\alpha \models \beta$ si y sólo si la sentencia $(\alpha \Rightarrow \beta)$ es válida.

(En el Ejercicio 7.4 se pide demostrar una serie de aserciones.) Podemos pensar en el algoritmo de inferencia de la Figura 7.10 como en un proceso para averiguar la validez de $(BC \Rightarrow \alpha)$. A la inversa, cada sentencia que es una implicación válida representa una inferencia correcta.

SATISFACIBILIDAD

El último concepto que necesitaremos es el de ***satisfacibilidad***. Una sentencia es satisfactoria si es verdadera para *algún* modelo. Por ejemplo, en la base de conocimiento ya mostrada, $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$ es *satisfacible* porque hay tres modelos en los que es verdadera, tal como se muestra en la Figura 7.9. Si una sentencia α es verdadera en un modelo m, entonces decimos que m **satisface** α, o que m **es un modelo de** α. La *satisfacibilidad* se puede averiguar enumerando los modelos posibles hasta que uno satisface la sentencia. La determinación de la *satisfacibilidad* de sentencias en lógica proposicional fue el primer problema que se demostró que era NP-completo.

SATISFACE

Muchos problemas en las ciencias de la computación son en realidad problemas de *satisfacibilidad*. Por ejemplo, todos los problemas de satisfacción de restricciones del Capítulo 5 se preguntan esencialmente si un conjunto de restricciones se satisfacen dada una asignación. Con algunas transformaciones adecuadas, los problemas de búsqueda también se pueden resolver mediante *satisfacibilidad*. La validez y la *satisfacible* están íntimamente relacionadas: α es válida si y sólo si $\neg\alpha$ es *insatisfacible*; en contraposición, α es *satisfacible* si y sólo si $\neg\alpha$ no es válida.

$\alpha \models \beta$ si y sólo si la sentencia $(\alpha \wedge \neg\beta)$ es insatisfactoria.

REDUCTIO AD ABSURDUM

REFUTACIÓN

La demostración de β a partir de α averiguando la insatisfacibilidad de $(\alpha \wedge \neg\beta)$ se corresponde exactamente con la técnica de demostración en matemáticas de la *reductio ad absurdum* (que literalmente se traduce como «reducción al absurdo»). Esta técnica también se denomina demostración mediante **refutación** o demostración por **contradicción**. Asumimos que la sentencia β es falsa y observamos si se llega a una contradicción con las premisas en α. Dicha contradicción es justamente lo que queremos expresar cuando decimos que la sentencia $(\alpha \wedge \neg\beta)$ es *insatisfacible*.

7.5 Patrones de razonamiento en lógica proposicional

Esta sección cubre los patrones estándar de inferencia que se pueden aplicar para derivar cadenas de conclusiones que nos llevan al objetivo deseado. Estos patrones de infe-

REGLAS DE INFERENCIA

MODUS PONENS

rencia se denominan **reglas de inferencia**. La regla más conocida es la llamada **Modus Ponens** que se escribe como sigue:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

La notación nos dice que, cada vez que encontramos dos sentencias en la forma $\alpha \Rightarrow \beta$ y α, entonces la sentencia β puede ser inferida. Por ejemplo, si tenemos (*WumpusEnFrente* $\wedge$ *WumpusVivo*) $\Rightarrow$ *Disparar* y (*WumpusEnFrente* $\wedge$ *WumpusVivo*), entonces se puede inferir *Disparar*.

ELIMINACIÓN-$\wedge$

Otra regla de inferencia útil es la **Eliminación-$\wedge$**, que expresa que, de una conjunción se puede inferir cualquiera de sus conjuntores:

$$\frac{\alpha \wedge \beta}{\alpha}$$

Por ejemplo, de (*WumpusEnFrente* $\wedge$ *WumpusVivo*), se puede inferir *WumpusVivo*.

Teniendo en cuenta los posibles valores de verdad de α y β se puede observar fácilmente, de una sola vez, que el Modus Ponens y la Eliminación-$\wedge$ son reglas sólidas. Estas reglas se pueden utilizar sobre cualquier instancia en la que es aplicable, generando inferencias sólidas, sin la necesidad de enumerar todos los modelos.

Todas las equivalencias lógicas de la Figura 7.11 se pueden utilizar como reglas de inferencia. Por ejemplo, la equivalencia de la eliminación de la bicondicional nos lleva a las dos reglas de inferencia

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{y} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Pero no todas las reglas de inferencia se pueden usar, como ésta, en ambas direcciones. Por ejemplo, no podemos utilizar el Modus Ponens en la dirección opuesta para obtener $\alpha \Rightarrow \beta$ y α a partir de β.

Veamos cómo se pueden usar estas reglas de inferencia y equivalencias en el mundo de *wumpus*. Comenzamos con la base de conocimiento conteniendo R_1 a R_5, y mostramos cómo demostrar $\neg H_{1,2}$, es decir, que no hay un hoyo en la casilla [1, 2]. Primero aplicamos la eliminación de la bicondicional a R_2 para obtener

$$R_6: \quad (B_{1,1} \Rightarrow (H_{1,2} \vee H_{2,1})) \wedge ((H_{1,2} \vee H_{2,1}) \Rightarrow \mathrm{B}_{1,1})$$

Entonces aplicamos la Eliminación-$\wedge$ a R_6 para obtener

$$R_7: \quad ((H_{1,2} \vee H_{2,1}) \Rightarrow B_{1,1})$$

Y por la equivalencia lógica de contraposición obtenemos

$$R_8: \quad (\neg B_{1,1} \Rightarrow \neg(H_{1,2} \vee H_{2,1}))$$

Ahora aplicamos el Modus Ponens con R_8 y la percepción R_4 (por ejemplo, $\neg \mathrm{B}_{1,1}$), para obtener

$$R_9: \quad \neg(H_{1,2} \vee H_{2,1})$$

Finalmente, aplicamos la ley de Morgan, obteniendo la conclusión

$$R_{10}: \neg H_{1,2} \wedge \neg H_{2,1}$$

Es decir, ni la casilla [1, 2] ni la [2, 1] contienen un hoyo.

PRUEBA

A la derivación que hemos realizado (una secuencia de aplicaciones de reglas de inferencia) se le denomina una **prueba** (o **demostración**). Obtener una prueba es muy semejante a encontrar una solución en un problema de búsqueda. De hecho, si la función sucesor se define para generar todas las aplicaciones posibles de las reglas de inferencia, entonces todos los algoritmos de búsqueda del Capítulo 3 se pueden utilizar para obtener una prueba. De esta manera, la búsqueda de pruebas es una alternativa a tener que enumerar los modelos. La búsqueda se puede realizar hacia delante a partir de la base de conocimiento inicial, aplicando las reglas de inferencia para derivar la sentencia objetivo, o hacia atrás, desde la sentencia objetivo, intentando encontrar una cadena de reglas de inferencia que nos lleven a la base de conocimiento inicial. Más adelante, en esta sección, veremos dos familias de algoritmos que utilizan estas técnicas.

El hecho de que la inferencia en lógica proposicional sea un problema NP-completo nos hace pensar que, en el peor de los casos, la búsqueda de pruebas va a ser no mucho más eficiente que la enumeración de modelos. Sin embargo, en muchos casos prácticos, *encontrar una prueba puede ser altamente eficiente simplemente porque el proceso puede ignorar las proposiciones irrelevantes, sin importar cuántas de éstas haya*. Por ejemplo, la prueba que hemos visto que nos llevaba a $\neg H_{1,2} \wedge \neg H_{2,1}$ no utiliza las proposiciones $B_{2,1}$, $H_{1,1}$, $H_{2,2}$ o $H_{3,1}$. Estas proposiciones se pueden ignorar porque la proposición objetivo $H_{1,2}$ sólo aparece en la sentencia R_4, y la otra proposición de R_4 sólo aparece también en R_2; por lo tanto, R_1, R_3 y R_5 no juegan ningún papel en la prueba. Sucedería lo mismo aunque añadiésemos un millón de sentencias a la base de conocimiento; por el otro lado, el algoritmo de la tabla de verdad, aunque sencillo, quedaría saturado por la explosión exponencial de los modelos.

MONÓTONO

Esta propiedad de los sistemas lógicos en realidad proviene de una característica mucho más fundamental, denominada **monótono**. La característica de monotonismo nos dice que el conjunto de sentencias implicadas sólo puede *aumentar* (pero no cambiar) al añadirse información a la base de conocimiento[10]. Para cualquier sentencia α y β,

$$\text{si} \quad BC \models \alpha \quad \text{entonces} \quad BC \wedge \beta \models \alpha$$

Por ejemplo, supongamos que la base de conocimiento contiene una aserción adicional β, que nos dice que hay exactamente ocho hoyos en el escenario. Este conocimiento podría ayudar al agente a obtener conclusiones *adicionales*, pero no puede invalidar ninguna conclusión α ya inferida (como la conclusión de que no hay un hoyo en la casilla [1, 2]). El monotonismo permite que las reglas de inferencia se puedan aplicar siempre que se hallen premisas aplicables en la base de conocimiento; la conclusión de la regla debe permanecer *sin hacer caso de qué más hay en la base de conocimiento*.

[10] Las lógicas **No Monótonas**, que violan la propiedad de monotonismo, modelan una característica propia del razonamiento humano: cambiar de opinión. Estas lógicas se verán en la Sección 10.7.

Resolución

Hemos argumentado que las reglas de inferencia vistas hasta aquí son *sólidas*, pero no hemos visto la cuestión acerca de lo *completo* de los algoritmos de inferencia que las utilizan. Los algoritmos de búsqueda como el de búsqueda en profundidad iterativa (página 87) son completos en el sentido de que éstos encontrarán cualquier objetivo alcanzable, pero si las reglas de inferencia no son adecuadas, entonces el objetivo no es alcanzable; no existe una prueba que utilice sólo esas reglas de inferencia. Por ejemplo, si suprimimos la regla de eliminación de la bicondicional la prueba de la sección anterior no avanzaría. En esta sección se introduce una regla de inferencia sencilla, la **resolución**, que nos lleva a un algoritmo de inferencia completo cuando se empareja a un algoritmo de búsqueda completo.

Comenzaremos utilizando una versión sencilla de la resolución aplicada al mundo de *wumpus*. Consideremos los pasos que nos llevaban a la Figura 7.4(a): el agente vuelve de la casilla [2, 1] a la [1, 1] y entonces va a la casilla [1, 2], donde percibe un hedor, pero no percibe una corriente de aire. Ahora añadimos los siguientes hechos a la base de conocimiento:

$$R_{11}: \quad \neg B_{1,2}$$
$$R_{12}: \quad B_{1,2} \Leftrightarrow (H_{1,1} \vee H_{2,2} \vee H_{1,3})$$

Mediante el mismo proceso que nos llevó antes a R_{10}, podemos derivar que no hay ningún hoyo en la casilla [2, 2] o en la [1, 3] (recuerde que se sabe que en la casilla no había ninguna percepción de hoyos):

$$R_{13}: \quad \neg \mathrm{H}_{2,2}$$
$$R_{14}: \quad \neg \mathrm{H}_{1,3}$$

También podemos aplicar la eliminación de la bicondicional a la R_3, seguido del Modus Ponens con la R_5, para obtener el hecho de que puede haber un hoyo en la casilla [1, 1], la [2, 2] o la [3, 1]:

$$R_{15}: \quad \mathrm{H}_{1,1} \vee \mathrm{H}_{2,2} \vee \mathrm{H}_{3,1}$$

Ahora viene la primera aplicación de la regla de resolución: el literal $\neg \mathrm{H}_{2,2}$ de la R_{13} *se resuelve con* el literal $\mathrm{H}_{2,2}$ de la R_{15} , dando el *resolvente*

$$R_{16}: \quad \mathrm{H}_{1,1} \vee \mathrm{H}_{3,1}$$

En lenguaje natural: si hay un hoyo en la casilla [1, 1], o en la [2, 2], o en la [3, 1], y no hay ninguno en la [2, 2], entonces hay uno en la [1, 1] o en la [3, 1]. De forma parecida, el literal $\neg \mathrm{H}_{1,1}$ de la R_1 se resuelve con el literal $\mathrm{H}_{1,1}$ de la R_{16}, dando

$$R_{17}: \quad \mathrm{H}_{3,1}$$

En lenguaje natural: si hay un hoyo en la casilla [1, 1] o en la [3, 1], y no hay ninguno en la [1, 1], entonces hay uno en la [3, 1]. Los últimos dos pasos de inferencia son ejemplo de la regla de inferencia de **resolución unitaria**,

RESOLUCIÓN UNITARIA

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

LITERALES COMPLEMENTARIOS

CLÁUSULA

CLÁUSULA UNITARIA

en donde cada ℓ es un literal y ℓ_i y m son **literales complementarios** (por ejemplo, uno es la negación del otro). Así, la resolución unitaria toma una **cláusula** (una disyunción de literales) y un literal para producir una nueva cláusula. Fíjese en que un literal se puede ver como una disyunción con un solo literal, conocido como **cláusula unitaria**.

La regla de resolución unitaria se puede generalizar a la regla general de **resolución**,

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

donde ℓ_i y m_j son literales complementarios. Si sólo tratáramos con cláusulas de longitud dos, podríamos escribir la regla así

$$\frac{\ell_1 \vee \ell_2, \qquad \neg\ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Es decir, la resolución toma dos cláusulas y genera una cláusula nueva con los literales de las dos cláusulas originales *menos* los literales complementarios. Por ejemplo, tendríamos

$$\frac{P_{1,1} \vee P_{3,1}, \qquad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

FACTORIZACIÓN

Hay otro aspecto técnico relacionado con la regla de resolución: la cláusula resultante debería contener sólo una copia de cada literal[11]. Se le llama **factorización** al proceso de eliminar las copias múltiples de los literales. Por ejemplo, si resolvemos $(A \vee B)$ con $(A \vee \neg B)$ obtenemos $(A \vee A)$, que se reduce a A.

La *solidez* de la regla de resolución se puede ver fácilmente si consideramos el literal ℓ_i. Si ℓ_i es verdadero, entonces m_j es falso, y de aquí $m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n$ debe ser verdadero, porque se da $m_1 \vee \cdots \vee m_n$. Si ℓ_i es falso, entonces $\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k$ debe ser verdadero, porque se da $\ell_1 \vee \cdots \vee \ell_k$. Entonces l_i es o bien verdadero o bien falso, y así, se obtiene una de las dos conclusiones, exactamente tal cómo establece la regla de resolución.

Lo que es más sorprendente de la regla de resolución es que crea la base para una familia de procedimientos de inferencia *completos*. *Cualquier algoritmo de búsqueda completo, aplicando sólo la regla de resolución, puede derivar cualquier conclusión implicada por cualquier base de conocimiento en lógica proposicional*. Pero hay una advertencia: la resolución es completa en un sentido muy especializado. Dado que A sea

[11] Si una cláusula se ve como un conjunto de literales, entonces esta restricción se respeta de forma automática. Utilizar la notación de conjuntos para representar cláusulas hace que la regla de resolución sea más clara, con el coste de introducir una notación adicional.

verdadero, no podemos utilizar la resolución para generar de forma automática la consecuencia $A \vee B$. Sin embargo, podemos utilizar la resolución para responder a la pregunta de si $A \vee B$ es verdadero. Este hecho se denomina **completitud de la resolución**, que indica que la resolución se puede utilizar siempre para confirmar o refutar una sentencia, pero no se puede usar para enumerar sentencias verdaderas. En las dos siguientes secciones explicamos cómo la resolución lleva a cabo este proceso.

COMPLETITUD DE LA RESOLUCIÓN

Forma normal conjuntiva

La regla de resolución sólo se puede aplicar a disyunciones de literales, por lo tanto, sería muy importante que la base de conocimiento y las preguntas a ésta estén formadas por disyunciones. Entonces, ¿cómo nos lleva esto a un procedimiento de inferencia completa para toda la lógica proposicional? La respuesta es que *toda sentencia en lógica proposicional es equivalente lógicamente a una conjunción de disyunciones de literales*. Una sentencia representada mediante una conjunción de disyunciones de literales se dice que está en **forma normal conjuntiva** o **FNC**. Que lo consideraremos bastante útil más tarde, al tratar la reducida familia de sentencias ***k*-FNC**. Una sentencia k-FNC tiene exactamente k literales por cláusula:

FORMA NORMAL CONJUNTIVA

K-FNC

$$(\ell_{1,1} \vee \ldots \vee \ell_{1,k}) \wedge \ldots \wedge (\ell_{n,1} \vee \ldots \vee \ell_{n,k})$$

De manera que se puede transformar cada sentencia en una sentencia de tipo 3-FNC, la cual tiene un conjunto de modelos equivalente.

Mejor que demostrar estas afirmaciones (*véase* el Ejercicio 7.10), vamos a describir un procedimiento de conversión muy sencillo. Vamos a ilustrar el procedimiento con la conversión de R_2, la sentencia $B_{1,1} \Leftrightarrow (H_{1,2} \vee H_{2,1})$, a FNC. Los pasos a seguir son los siguientes:

1. Eliminar $\Leftrightarrow$, sustituyendo $\alpha \Leftrightarrow \beta$ por $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (H_{1,2} \vee H_{2,1})) \wedge ((H_{1,2} \vee H_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminar $\Rightarrow$, sustituyendo $\alpha \Rightarrow \beta$ por $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge (\neg(H_{1,2} \vee H_{2,1}) \vee B_{1,1})$$

3. Una FNC requiere que la $\neg$ se aplique sólo a los literales, por lo tanto, debemos «anidar las $\neg$» mediante la aplicación reiterada de las siguientes equivalencias (sacadas de la Figura 7.11).

$$\neg(\neg\alpha) \equiv \alpha \text{ (eliminación de la doble negación)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \text{ (de Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \text{ (de Morgan)}$$

En el ejemplo, sólo necesitamos una aplicación de la última regla:

$$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge ((\neg H_{1,2} \wedge \neg H_{2,1}) \vee B_{1,1})$$

4. Ahora tenemos una sentencia que tiene una $\wedge$ con operadores de $\vee$ anidados, aplicados a literales y a una $\wedge$ anidada. Aplicamos la ley de distributividad de la Figura 7.11, distribuyendo la $\vee$ sobre la $\wedge$ cuando nos es posible.

$$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge (\neg H_{1,2} \vee B_{1,1}) \wedge (\neg H_{2,1} \vee \mathrm{B}_{1,1})$$

La sentencia inicial ahora está en FNC, una conjunción con tres cláusulas. Es más difícil de leer pero se puede utilizar como entrada en el procedimiento de resolución.

Un algoritmo de resolución

Los procedimientos de inferencia basados en la resolución trabajan utilizando el principio de prueba mediante contradicción que vimos al final de la Sección 7.4. Es decir, para demostrar que $BC \models \alpha$, demostramos que $(BC \wedge \neg\alpha)$ es *insatisfacible*. Lo hacemos demostrando una contradicción.

En la Figura 7.12 se muestra un algoritmo de resolución. Primero se convierte $(BC \wedge \neg\alpha)$ a FNC. Entonces, se aplica la regla de resolución a las cláusulas obtenidas. Cada par que contiene literales complementarios se resuelve para generar una nueva cláusula, que se añade al conjunto de cláusulas si no estaba ya presente. El proceso continúa hasta que sucede una de estas dos cosas:

- No hay nuevas cláusulas que se puedan añadir, en cuyo caso α no implica β , o
- Se deriva la cláusula vacía de una aplicación de la regla de resolución, en cuyo caso α implica β.

La cláusula vacía (una disyunción sin disyuntores) es equivalente a *Falso* porque una disyunción es verdadera sólo si al menos uno de sus disyuntores es verdadero. Otra forma de ver que la cláusula vacía representa una contradicción es observar que se presenta sólo si se resuelven dos cláusulas unitarias complementarias, tales como P y ¬P.

```
función RESOLUCIÓN-LP(BC, α) devuelve verdadero o falso
  entradas: BC, la base de conocimiento, una sentencia en lógica proposicional
            α, la petición, una sentencia en lógica proposicional

  cláusulas ← el conjunto de cláusulas de BC ∧ ¬α en representación FNC
  nueva ← { }
  bucle hacer
     para cada Ci, Cj en cláusulas hacer
        resolventes ← RESUELVE-LP(Ci, Cj)
        si resolventes contiene la cláusula vacía entonces devolver verdadero
        nueva ← nueva ∪ resolventes
     si nueva ⊆ cláusulas entonces devolver falso
     cláusulas ← cláusulas ∪ nueva
```

Figura 7.12 Un algoritmo sencillo de resolución para la lógica proposicional. La función RESUELVE-LP devuelve el conjunto de todas las cláusulas posibles que se obtienen de resolver las dos entradas.

Ahora podemos aplicar el procedimiento de resolución a una inferencia sencilla del mundo de *wumpus*. Cuando el agente está en la casilla [1, 1] no percibe ninguna brisa, por lo tanto no puede haber hoyos en las casillas vecinas. Las sentencias relevantes en la base de conocimiento son

$$BC = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (H_{1,2} \vee H_{2,1})) \wedge \neg B_{1,1}$$

y deseamos demostrar α, es decir $\neg H_{1,2}$. Cuando convertimos $(BC \wedge \neg\alpha)$ a FNC obtenemos las cláusulas que se muestran en la fila superior de la Figura 7.13. La segunda fila en la figura muestra todas las cláusulas obtenidas resolviendo parejas de la primera fila. Entonces, cuando $H_{1,2}$ se resuelve con $\neg H_{1,2}$ obtenemos la cláusula vacía, representada mediante un cuadrado pequeño. Una revisión de la Figura 7.13 nos revela que muchos pasos de resolución no nos sirven de nada. Por ejemplo, la cláusula $B_{1,1} \vee \neg B_{1,1} \vee H_{1,2}$ es equivalente a *Verdadero* $\vee H_{1,2}$, que es también equivalente a *Verdadero*. Deducir que *Verdadero* es verdadero no nos es muy útil. Por lo tanto, se puede descartar cualquier cláusula que contenga dos literales complementarios.

Completitud de la resolución

CIERRE DE LA RESOLUCIÓN

Para concluir con nuestro debate acerca de la resolución, ahora vamos a demostrar por qué es completo el procedimiento RESOLUCIÓN-LP. Para hacerlo nos vendrá bien introducir el concepto de **cierre de la resolución** $CR(S)$ del conjunto de cláusulas S, que es el conjunto de todas las cláusulas derivables, obtenidas mediante la aplicación repetida de la regla de resolución a las cláusulas de S o a las derivadas de éstas. El cierre de la resolución es lo que calcula el procedimiento RESOLUCIÓN-LP y asigna como valor final a la variable *cláusulas*. Es fácil ver que $CR(S)$ debe ser finito, porque sólo hay un conjunto finito de las diferentes cláusulas que se pueden generar a partir del conjunto de símbolos $P_1 \ldots, P_k$ que aparecen en S, (fíjese que esto no sería cierto si no aplicáramos el procedimiento de factorización, que elimina las copias múltiples de un literal). Por eso, el procedimiento RESOLUCIÓN-LP siempre termina.

Figura 7.13 Aplicación parcial de RESOLUCIÓN-LP a una inferencia sencilla en el mundo de wumpus. Se observa que $\neg H_{1,2}$ se sigue de las cláusulas 3.ª y 4.ª de la fila superior.

TEOREMA FUNDAMENTAL DE LA RESOLUCIÓN

El teorema de la completitud para la resolución en lógica proposicional se denomina **teorema fundamental de la resolución**:

> Si un conjunto de cláusulas es *insatisfacible*, entonces el cierre de la resolución de esas cláusulas contiene la cláusula vacía.

Vamos a probar este teorema demostrando su contraposición: si el cierre $CR(S)$ *no* contiene la cláusula vacía, entonces S es *satisfacible*. De hecho, podemos construir un modelo de S con los valores de verdad adecuados para $P_1 \ldots, P_k$. El procedimiento de construcción es como sigue:

Para i de 1 a k,

— Si hay una cláusula en $CR(S)$ que contenga el literal $\neg P_i$, tal que todos los demás literales de la cláusula sean falsos bajo la asignación escogida para $P_1 \ldots, P_{i-1}$, entonces asignar a P_i el valor de *falso*.
— En otro caso, asignar a P_i el valor de *verdadero*.

Queda por demostrar que esta asignación a $P_1 \ldots, P_k$ es un modelo de S, a condición de que $CR(S)$ se cierre bajo la resolución y no contenga la cláusula vacía. Esta demostración se deja como ejercicio.

Encadenamiento hacia delante y hacia atrás

La completitud de la resolución hace que ésta sea un método de inferencia muy importante. Sin embargo, en muchos casos prácticos no se necesita todo el poder de la resolución. Las bases de conocimiento en el mundo real a menudo contienen sólo cláusulas, de un tipo restringido, denominadas **cláusulas de Horn**. Una cláusula de Horn es una disyunción de literales de los cuales, *como mucho uno es positivo*. Por ejemplo, la cláusula $(\neg L_{1,1} \vee \neg Brisa \vee B_{1,1})$, en donde $L_{1,1}$ representa que el agente está en la casilla [1, 1], es una cláusula de Horn, mientras que la cláusula $(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1})$ no lo es.

CLÁUSULAS DE HORN

La restricción de que haya sólo un literal positivo puede parecer algo arbitraria y sin interés, pero realmente es muy importante, debido a tres razones:

1. Cada cláusula de Horn se puede escribir como una implicación cuya premisa sea una conjunción de literales positivos y cuya conclusión sea un único literal positivo. (*Véase* el Ejercicio 7.12.) Por ejemplo, la cláusula de Horn $(\neg L_{1,1} \vee \neg Brisa \vee B_{1,1})$ se puede rescribir como la implicación $(L_{1,1} \wedge Brisa) \Rightarrow B_{1,1}$. La sentencia es más fácil de leer en la última representación: ésta dice que si el agente está en la casilla [1, 1] y percibe una brisa, entonces la casilla [1, 1] tiene una corriente de aire. La gente encuentra más fácil esta forma de leer y escribir sentencias para muchos dominios del conocimiento.

   Las cláusulas de Horn como ésta, con *exactamente* un literal positivo, se denominan **cláusulas positivas**. El literal positivo se denomina **cabeza**, y la disyunción de literales negativos **cuerpo** de la cláusula. Una cláusula positiva que no tiene literales negativos simplemente aserta una proposición dada, que algunas veces se le denomina **hecho**. Las cláusulas positivas forman la base de la

CLÁUSULAS POSITIVAS

CABEZA

CUERPO

HECHO

programación lógica, que se verá en el Capítulo 9. Una cláusula de Horn *sin* literales positivos se puede escribir como una implicación cuya conclusión es el literal *Falso*. Por ejemplo, la cláusula $(\neg W_{1,1} \vee \neg W_{1,2})$ (el *wumpus* no puede estar en la casilla [1, 1] y la [1, 2] a la vez) es equivalente a $W_{1,1} \wedge W_{1,2} \Rightarrow$ *Falso*. A este tipo de sentencias se las llama **restricciones de integridad** en el mundo de las bases de datos, donde se utilizan para indicar errores entre los datos. En los algoritmos siguientes asumimos, para simplificar, que la base de conocimiento sólo contiene cláusulas positivas y que no dispone de restricciones de integridad. Entonces decimos que estas bases de conocimiento están en forma de Horn.

RESTRICCIONES DE INTEGRIDAD

2. La inferencia con cláusulas de Horn se puede realizar mediante los algoritmos de **encadenamiento hacia delante** y de **encadenamiento hacia atrás**, que en breve explicaremos. Ambos algoritmos son muy naturales, en el sentido de que los pasos de inferencia son obvios y fáciles de seguir por las personas.

ENCADENAMIENTO HACIA DELANTE

ENCADENAMIENTO HACIA ATRÁS

3. Averiguar si hay o no implicación con las cláusulas de Horn se puede realizar en un tiempo que es *lineal* respecto al tamaño de la base de conocimiento.

Este último hecho es una grata sorpresa. Esto significa que la inferencia lógica es un proceso barato para muchas bases de conocimiento en lógica proposicional que se encuentran en el mundo real.

El algoritmo de encadenamiento hacia delante ¿IMPLICACIÓN-EHD-LP?(*BC*, *q*) determina si un símbolo proposicional *q* (la petición) se deduce de una base de conocimiento compuesta por cláusulas de Horn. El algoritmo comienza a partir de los hechos conocidos (literales positivos) de la base de conocimiento. Si todas las premisas de una implicación se conocen, entonces la conclusión se añade al conjunto de hechos conocidos. Por ejemplo, si $L_{1,1}$ y *Brisa* se conocen y $(L_{1,1} \wedge Brisa) \Rightarrow B_{1,1}$ está en la base de conocimiento, entonces se puede añadir $B_{1,1}$ a ésta. Este proceso continúa hasta que la petición *q* es añadida o hasta que no se pueden realizar más inferencias. En la Figura 7.14 se muestra el algoritmo detallado. El principal punto a recordar es que el algoritmo se ejecuta en tiempo lineal.

La mejor manera de entender el algoritmo es mediante un ejemplo y un diagrama. La Figura 7.15(a) muestra una base de conocimiento sencilla con cláusulas de Horn, en donde *A* y *B* se conocen como hechos. La Figura 7.15(b) muestra la misma base de conocimiento representada mediante un **grafo Y-O**. En los grafos Y-O múltiples enlaces se juntan mediante un arco para indicar una disyunción (cualquier enlace se puede probar). Es fácil ver cómo el encadenamiento hacia delante trabaja sobre el grafo. Se seleccionan los hechos conocidos (aquí *A* y *B*) y la inferencia se propaga hacia arriba tanto como se pueda. Siempre que aparece una conjunción, la propagación se para hasta que todos los conjuntores sean conocidos para seguir a continuación. Se anima al lector a que desarrolle el proceso en detalle a partir del ejemplo.

GRAFO Y-O

Es fácil descubrir que el encadenamiento hacia delante es un proceso **sólido**: cada inferencia es esencialmente una aplicación del Modus Ponens. El encadenamiento hacia delante también es **completo**: cada sentencia atómica implicada será derivada. La forma más fácil de verlo es considerando el estado final de la tabla *inferido* (después de que el algoritmo encuentra un **punto fijo** a partir del cual no es posible realizar nuevas inferencias).

PUNTO FIJO

```
función ¿IMPLICACIÓN-EHD-LP?(BC, q) devuelve verdadero o falso
  entradas: BC, la base de conocimiento, un conjunto de cláusulas de Horn en Lógica Proposicional
            q, la petición, un símbolo proposicional
  variables locales: cuenta, una tabla ordenada por cláusula, inicializada al número de cláusulas
                     inferido, una tabla, ordenada por símbolo, cada entrada inicializada a falso
                     agenda, una lista de símbolos, inicializada con los símbolos de la BC que se
                     sabe que son verdaderos

  mientras agenda no esté vacía hacer
    p ← POP(agenda)
    a menos que inferido[p] hacer
        inferido[p] ← verdadero
        para cada cláusula de Horn c en la que aparezca la premisa p hacer
            reducir cuenta[c]
            si cuenta[c] = 0 entoncer hacer
                si CABEZA[c] = q entonces devolver verdadero
                PUSH(CABEZA[c], agenda)
  devolver falso
```

Figura 7.14 El algoritmo de encadenamiento hacia delante para la lógica proposicional. La variable *agenda* almacena la pista de los símbolos que se saben son verdaderos pero no han sido «procesados» todavía. La tabla *cuenta* guarda la pista de las premisas de cada implicación que aún son desconocidas. Siempre que se procesa un símbolo p de la agenda la cuenta se reduce en uno para cada implicación en la que aparece la premisa p. (Este proceso se puede realizar en un tiempo constante si la *BC* se ordena de forma adecuada.) Si la cuenta llega a cero, todas las premisas de la implicación son conocidas, y por tanto, la conclusión se puede añadir a la agenda. Por último, necesitamos guardar la pista de qué símbolos han sido procesados; no se necesita añadir un símbolo inferido si ha sido procesado previamente. Este proceso nos evita un trabajo redundante, y también nos prevé de los bucles infinitos que podrían causarse por implicaciones tales como $P \Rightarrow Q$ y $Q \Rightarrow P$.

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

(a)

(b)

Figura 7.15 (a) Una base de conocimiento sencilla con cláusulas de Horn. (b) Su correspondiente grafo Y-O.

La tabla contiene el valor *verdadero* para cada símbolo inferido en el proceso, y el valor *falso* para los demás símbolos. Podemos interpretar la tabla como un modelo lógico, más aún, *cada cláusula positiva de la BC original es verdadera en este modelo*. Para ver esto asumamos lo opuesto, en concreto, que alguna cláusula $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ sea falsa en el modelo. Entonces $a_1 \wedge \ldots \wedge a_k$ debe ser verdadero en el modelo y b debe ser falso. ¡Pero esto contradice nuestra asunción de que el algoritmo ha encontrado un punto fijo! Por lo tanto, podemos concluir que el conjunto de sentencias atómicas inferidas hasta el punto fijo define un modelo de la *BC* original. Además, cualquier sentencia atómica q que se implica de la *BC* debe ser cierta en todos los modelos y en este modelo en particular. Por lo tanto, cada sentencia implicada q debe ser inferida por el algoritmo.

DIRIGIDO POR LOS DATOS

El encadenamiento hacia delante es un ejemplo del concepto general de razonamiento **dirigido por los datos**, es decir, un razonamiento en el que el foco de atención parte de los datos conocidos. Este razonamiento se puede utilizar en un agente para derivar conclusiones a partir de percepciones recibidas, a menudo, sin la necesidad de una petición concreta. Por ejemplo, el agente de *wumpus* podría DECIR sus percepciones a la base de conocimiento utilizando un algoritmo de encadenamiento hacia delante de tipo incremental, en el que los hechos se pueden añadir a la agenda para iniciar nuevas inferencias. A las personas, a medida que les llega nueva información, se les activa una gran cantidad de razonamiento dirigido por los datos. Por ejemplo, si estoy en casa y oigo que comienza a llover, podría sucederme que la merienda quede cancelada. Con todo esto, no será muy probable que el pétalo diecisieteavo de la rosa más alta del jardín de mi vecino se haya mojado. Las personas llevan a cabo un encadenamiento hacia delante con un control cuidadoso, a fin de no hundirse en consecuencias irrelevantes.

El algoritmo de encadenamiento hacia atrás, tal como sugiere su nombre, trabaja hacia atrás a partir de la petición. Si se sabe que la petición q es verdadera, entonces no se requiere realizar ningún trabajo. En el otro caso, el algoritmo encuentra aquellas implicaciones de la base de conocimiento de las que se concluye q. Si se puede probar que todas las premisas de una de esas implicaciones son verdaderas (mediante en encadenamiento hacia atrás), entonces q es verdadera. Cuando se aplica a la petición Q de la Figura 7.15, el algoritmo retrocede hacia abajo por el grafo hasta que encuentra un conjunto de hechos conocidos que forma la base de la demostración. El algoritmo detallado se deja como ejercicio. Al igual que en el encadenamiento hacia delante, una implementación eficiente se ejecuta en tiempo lineal.

RAZONAMIENTO DIRIGIDO POR EL OBJETIVO

El encadenamiento hacia atrás es un tipo de **razonamiento dirigido por el objetivo**. Este tipo de razonamiento es útil para responder a peticiones tales como «¿Qué debo hacer ahora?» y «¿Dónde están mis llaves?» A menudo, el coste del encadenamiento hacia atrás es *mucho menor* que el orden lineal respecto al tamaño de la base de conocimiento, porque el proceso sólo trabaja con los hechos relevantes. Por lo general, un agente debería repartir su trabajo entre el razonamiento hacia delante y el razonamiento hacia atrás, limitando el razonamiento hacia delante a la generación de los hechos que sea probable que sean relevantes para las peticiones, y éstas se resolverán mediante el encadenamiento hacia atrás.

7.6 Inferencia proposicional efectiva

En esta sección vamos a describir dos familias de algoritmos eficientes para la inferencia en lógica proposicional, basadas en la comprobación de modelos: un enfoque basado en la búsqueda con *backtracking*, y el otro en la búsqueda basada en la escalada de colina. Estos algoritmos forman parte de la «tecnología» de la lógica proposicional. Esta sección se puede tan sólo ojear en una primera lectura del capítulo.

Los algoritmos que vamos a describir se utilizan para la comprobación de la *satisfacibilidad*. Ya hemos mencionado la conexión entre encontrar un modelo satisfacible para una sentencia lógica y encontrar una solución para un problema de satisfacción de restricciones, entonces, quizá no sorprenda que las dos familias de algoritmos se asemejen bastante a los algoritmos con *backtracking* de la Sección 5.2 y a los algoritmos de búsqueda local de la Sección 5.3. Sin embargo, éstos son extremadamente importantes por su propio derecho, porque muchos problemas combinatorios en las ciencias de la computación se pueden reducir a la comprobación de la *satisfacibilidad* de una sentencia proposicional. Cualquier mejora en los algoritmos de *satisfacibilidad* presenta enormes consecuencias para nuestra habilidad de manejar la complejidad en general.

Un algoritmo completo con *backtracking* («vuelta atrás»)

ALGORITMO DE DAVIS Y PUTNAM

El primer algoritmo que vamos a tratar se le llama a menudo **algoritmo de Davis y Putnam**, después del artículo de gran influencia que escribieron Martin Davis y Hilary Putnam (1960). De hecho, el algoritmo es la versión descrita por Davis, Logemann y Loveland (1962), así que lo llamaremos DPLL, las iniciales de los cuatro autores. DPLL toma como entrada una sentencia en forma normal conjuntiva (un conjunto de cláusulas). Del mismo modo que la BÚSQUEDA-CON-BACKTRACKING e ¿IMPLICACIÓN-TV?, DPLL realiza una enumeración esencialmente recursiva, mediante el primero en profundidad, de los posibles modelos. El algoritmo incorpora tres mejoras al sencillo esquema del ¿IMPLICACIÓN-TV?

- *Terminación anticipada*: el algoritmo detecta si la sentencia debe ser verdadera o falsa, aun con un modelo completado parcialmente. Una cláusula es verdadera si *cualquier* literal es verdadero, aun si los otros literales todavía no tienen valores de verdad; así la sentencia, entendida como un todo, puede evaluarse como verdadera aun antes de que el modelo sea completado. Por ejemplo, la sentencia $(A \vee B) \wedge (A \vee C)$ es verdadera si A lo es, sin tener en cuenta los valores de B y C. De forma similar, una sentencia es falsa si *cualquier* cláusula lo es, caso que ocurre cuando cada uno de sus literales lo son. Del mismo modo, esto puede ocurrir mucho antes de que el modelo sea completado. La terminación anticipada evita la evaluación íntegra de los subárboles en el espacio de búsqueda.

SÍMBOLO PURO

- *Heurística de símbolo puro*: un **símbolo puro** es un símbolo que aparece siempre con el mismo «signo» en todas las cláusulas. Por ejemplo, en las tres cláusulas $(A \vee \neg B)$, $(\neg B \vee \neg C)$, y $(C \vee A)$, el símbolo A es puro, porque sólo aparece el literal positivo, B también es puro porque sólo aparece el literal negativo, y C es un sím-

bolo impuro. Es fácil observar que si una sentencia tiene un modelo, entonces tiene un modelo con símbolos puros asignados para hacer que sus literales sean *verdaderos*, porque al hacerse así una cláusula nunca puede ser falsa. Fíjese en que, al determinar la pureza de un símbolo, el algoritmo puede ignorar las cláusulas que ya se sabe que son verdaderas en el modelo construido hasta ahora. Por ejemplo, si el modelo contiene $B = falso$, entonces la cláusula $(\neg B \vee \neg C)$ ya es verdadera, y C pasa a ser un símbolo puro, ya que sólo aparecerá en la cláusula $(C \vee A)$.

- *Heurística de cláusula unitaria*: anteriormente había sido definida una **cláusula unitaria** como aquella que tiene sólo un literal. En el contexto del DPLL, este concepto también determina a aquellas cláusulas en las que todos los literales, menos uno, tienen asignado el valor *falso* en el modelo. Por ejemplo, si el modelo contiene $B = falso$, entonces $(B \vee \neg C)$ pasa a ser una cláusula unitaria, porque es equivalente a $(Falso \vee \neg C)$, o justamente $\neg C$. Obviamente, para que esta cláusula sea verdadera, a C se le debe asignar *falso*. La heurística de cláusula unitaria asigna dichos símbolos antes de realizar la ramificación restante. Una consecuencia importante de la heurística es que cualquier intento de demostrar (mediante refutación) un literal que ya esté en la base de conocimiento, tendrá éxito inmediatamente (Ejercicio 7.16). Fíjese también en que la asignación a una cláusula unitaria puede crear otra cláusula unitaria (por ejemplo, cuando a C se le asigna *falso*, $(C \vee A)$ pasa a ser una cláusula unitaria, causando que le sea asignado a A el valor verdadero). A esta «cascada» de asignaciones forzadas se la denomina **propagación unitaria**. Este proceso se asemeja al encadenamiento hacia delante con las cláusulas de Horn, y de hecho, si una expresión en FNC sólo contiene cláusulas de Horn, entonces el DPLL esencialmente reproduce el encadenamiento hacia delante. (*Véase* el Ejercicio 7.17.)

PROPAGACIÓN UNITARIA

En la Figura 7.16 se muestra el algoritmo DPLL. Sólo hemos puesto la estructura básica del algoritmo, que describe, en sí mismo, el proceso de búsqueda. No hemos descrito las estructuras de datos que se deben utilizar para hacer que cada paso de la búsqueda sea eficiente, ni los trucos que se pueden añadir para mejorar su comportamiento: aprendizaje de cláusulas, heurísticas para la selección de variables y reinicialización aleatoria. Cuando estas mejoras se añaden, el DPLL es uno de los algoritmos de *satisfacibilidad* más rápidos, a pesar de su antigüedad. La implementación CHAFF se utiliza para resolver problemas de verificación de *hardware* con un millón de variables.

Algoritmos de búsqueda local

Hasta ahora, en este libro hemos visto varios algoritmos de búsqueda local, incluyendo la ASCENSIÓN-DE-COLINA (página 126) y el TEMPLADO-SIMULADO (página 130). Estos algoritmos se pueden aplicar directamente a los problemas de *satisfacibilidad*, a condición de que elijamos la correcta función de evaluación. Como el objetivo es encontrar una asignación que satisfaga todas las cláusulas, una función de evaluación que cuente el número de clásulas *insatisfacibles* hará bien el trabajo. De hecho, ésta es exactamente la medida utilizada en el algoritmo MIN-CONFLICTOS para los PSR (página 170). Todos estos algoritmos realizan los pasos en el espacio de asignaciones completas,

```
función ¿SATISFACIBLE-DPLL?(s) devuelve verdadero o falso
  entradas: s, una sentencia en lógica proposicional

  cláusulas ← el conjunto de cláusulas de s en representación FNC
  símbolos ← una lista de los símbolos proposicionales de s
  devolver DPLL(cláusulas, símbolos, [ ])
```

```
función DPLL(cláusulas, símbolos, modelo) devuelve verdadero o falso

  si cada cláusula en cláusulas es verdadera en el modelo entonces devolver verdadero
  si alguna cláusula en cláusulas es falsa en el modelo entonces devolver falso
  P, valor ← ENCONTRAR-SÍMBOLO-PURO(símbolos, cláusulas, modelo)
  si P no está vacío entonces devolver
  DPLL(cláusulas, símbolos − P, EXTENDER(P, valor, modelo)
  P, valor ← ENCONTRAR-CLÁUSULA-UNITARIA(cláusulas, modelo)
  si P no está vacío entonces devolver
  DPLL(cláusulas, símbolos − P, EXTENDER(P, valor, modelo)
  P ← PRIMERO(símbolos); resto ← RESTO(símbolos)
  devolver DPLL(cláusulas, resto, EXTENDER(P, verdadero, modelo)) o
           DPLL(cláusulas, resto, EXTENDER(P, falso, modelo))
```

Figura 7.16 El algoritmo DPLL para la comprobación de la *satisfacibilidad* de una sentencia en lógica proposicional. En el texto se describen ENCONTRAR-SÍMBOLO-PURO y ENCONTRAR-CLÁUSULA-UNITARIA; cada una devuelve un símbolo (o ninguno) y un valor de verdad para asignar a dicho símbolo. Al igual que ¿IMPLICACIÓN-TV?, este algoritmo trabaja sobre modelos parciales.

intercambiando el valor de verdad de un símbolo a la vez. El espacio generalmente contiene muchos mínimos locales, requiriendo diversos métodos de aleatoriedad para escapar de ellos. En los últimos años se han realizado una gran cantidad de experimentos para encontrar un buen equilibrio entre la voracidad y la aleatoriedad.

Uno de los algoritmos más sencillos y eficientes que han surgido de todo este trabajo es el denominado SAT-CAMINAR (Figura 7.17). En cada iteración, el algoritmo selecciona una cláusula insatisfecha y un símbolo de la cláusula para intercambiarlo. El algoritmo escoge aleatoriamente entre dos métodos para seleccionar el símbolo a intercambiar: (1) un paso de «min-conflictos» que minimiza el número de cláusulas insatisfechas en el nuevo estado, y (2) un paso «pasada-aleatoria» que selecciona de forma aleatoria el símbolo.

¿El SAT-CAMINAR realmente trabaja bien? De forma clara, si el algoritmo devuelve un modelo, entonces la sentencia de entrada de hecho es *satisfacible*. ¿Qué sucede si el algoritmo devuelve *fallo*? En ese caso, no podemos decir si la sentencia es *insatisfacible* o si necesitamos darle más tiempo al algoritmo. Podríamos intentar asignarle a *max_intercambios* el valor infinito. En ese caso, es fácil ver que con el tiempo SAT-CAMINAR nos devolverá un modelo (si existe alguno) a condición de que la probabilidad $p > 0$. Esto es porque siempre hay una secuencia de intercambios que nos lleva a una asignación satisfactoria, y al final, los sucesivos pasos de movimientos aleatorios generarán dicha secuencia. Ahora bien, si *max_intercambios* es infinito y la sentencia es *insatisfacible*, entonces la ejecución del algoritmo nunca finalizará.

```
función SAT-CAMINAR(cláusulas, p, max_intercambios) devuelve un modelo satisfactorio o fallo
   entradas: cláusulas, un conjunto de cláusulas en lógica proposicional
             p, la probabilidad de escoger un movimiento «aleatorio», generalmente alrededor de 0,5
             max_intercambios el número de intercambios permitidos antes de abandonar

   modelo ← una asignación aleatoria de verdadero/falso a los símbolos de las cláusulas
   para i = 1 hasta max_intercambios hacer
      si modelo satisface las cláusulas entonces devolver modelo
      cláusula ← una cláusula seleccionada aleatoriamente de cláusulas que es falsa en el modelo
      con probabilidad p intercambia el valor en el modelo de un símbolo seleccionado
      aleatoriamente de la cláusula
      sino intercambia el valor de cualquier símbolo de la cláusula para que maximice el
      número de cláusulas satisfacibles
devolver fallo
```

Figura 7.17 El algoritmo SAT-CAMINAR para la comprobación de la *satisfacibilidad* mediante intercambio aleatorio de los valores de las variables. Existen muchas versiones de este algoritmo.

Lo que sugiere este problema es que los algoritmos de búsqueda local, como el SAT-CAMINAR, son más útiles cuando esperamos que haya una solución (por ejemplo, los problemas de los que hablamos en los Capítulos 3 y 5, por lo general, tienen solución). Por otro lado, los algoritmos de búsqueda local no detectan siempre la *insatisfacibilidad*, algo que se requiere para decidir si hay relación de implicación. Por ejemplo, un agente no puede utilizar la búsqueda local para demostrar, de forma fiable, si una casilla es segura en el mundo de *wumpus*. En lugar de ello, el agente puede decir «he pensado acerca de ello durante una hora y no he podido hallar ningún mundo posible en el que la casilla *no sea segura*». Si el algoritmo de búsqueda local es por lo general realmente más rápido para encontrar un modelo cuando éste existe, el agente se podría justificar asumiendo que el impedimento para encontrar un modelo indica *insatisfacibilidad*. Desde luego que esto no es lo mismo que una demostración, y el agente se lo debería pensar dos veces antes de apostar su vida en ello.

Problemas duros de *satisfacibilidad*

Vamos a ver ahora cómo trabaja en la práctica el DPLL. En concreto, estamos interesados en los problemas *duros* (o *complejos*), porque los problemas *fáciles* se pueden resolver con cualquier algoritmo antiguo. En el Capítulo 5 hicimos algunos descubrimientos sorprendentes acerca de cierto tipo de problemas. Por ejemplo, el problema de las *n*-reinas (pensado como un problema absolutamente difícil para los algoritmos de búsqueda con *backtracking*) resultó ser trivialmente sencillo para los métodos de búsqueda local, como el min-conflictos. Esto es a causa de que las soluciones están distribuidas muy densamente en el espacio de asignaciones, y está garantizado que cualquier asignación inicial tenga cerca una solución. Así, el problema de las *n*-reinas es sencillo porque está **bajo restricciones**.

BAJO RESTRICCIONES

Cuando observamos los problemas de *satisfacibilidad* en forma normal conjuntiva, un problema bajo restricciones es aquel que tiene relativamente *pocas* cláusulas res-

tringiendo las variables. Por ejemplo, aquí tenemos una sentencia en FNC-3 con cinco símbolos, y cinco cláusulas generadas aleatoriamente[12]:

$$(\neg D \vee \neg B \vee C) \wedge (D \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$$

16 de las 32 posibles asignaciones son modelos de esta sentencia, por lo tanto, de media, sólo se requerirían dos pasos para encontrar un modelo.

Entonces, ¿dónde se encuentran los problemas duros? Presumiblemente, si *incrementamos* el número de cláusulas, manteniendo fijo el número de símbolos, hacemos que el problema esté más restringido, y que sea más difícil encontrar las soluciones. Permitamos que m sea el número de cláusulas y n el número de símbolos. La Figura 7.18(a) muestra la probabilidad de que una sentencia en FNC-3 sea *satisfacible*, como una función de la relación cláusula/símbolo, m/n, con n fijado a 50. Tal como esperábamos, para una m/n pequeña, la probabilidad se acerca a 1, y para una m/n grande, la probabilidad se acerca a 0. La probabilidad cae de forma bastante brusca alrededor del valor de $m/n = 4{,}3$. Las sentencias en FNC que están cerca de este **punto crítico** se podrían definir como «casi *satisfacibles*» o «casi *insatisfacibles*». ¿Es en este punto donde encontramos los problemas duros?

PUNTO CRÍTICO

La Figura 7.18(b) muestra los tiempos de ejecución de los algoritmos DPLL y SAT-CAMINAR alrededor de este punto crítico, en donde hemos restringido nuestra atención sólo a los problemas *satisfacibles*. Tres cosas están claras: primero, los problemas que están cerca del punto crítico son *mucho* más difíciles que los otros problemas aleatorios. Segundo, aun con los problemas más duros, el DPLL es bastante efectivo (una media de unos pocos miles de pasos comparados con $2^{50} \approx 10^{15}$ en la enumeración de tablas de verdad). Tercero, SAT-CAMINAR es mucho más rápido que el DPLL en todo el rango.

Figura 7.18 (a) Gráfico que muestra la probabilidad de que una sentencia en FNC-3 con $n = 50$ símbolos sea *satisfacible*, en función del ratio cláusula/símbolo m/n. (b) Gráfico del tiempo de ejecución promedio del DPLL y del SAT-CAMINAR sobre 100 sentencias en FNC-3 aleatorias con $n = 50$, para un rango reducido de valores de m/n alrededor del punto crítico.

[12] Cada cláusula contiene tres símbolos *diferentes* seleccionados aleatoriamente, cada uno de ellos negado con una probabilidad del 50%.

Por supuesto, estos resultados sólo son para los problemas generados aleatoriamente. Los problemas reales no tienen necesariamente la misma estructura (en términos de proporciones entre literales positivos y negativos, densidades de conexiones entre cláusulas, etcétera) que los problemas aleatorios. Pero todavía en la práctica, el SAT-CAMINAR y otros algoritmos de la misma familia son muy buenos para resolver problemas reales (a menudo, tan buenos como el mejor algoritmo de propósito específico para esas tareas). Problemas con miles de símbolos y millones de cláusulas se tratan de forma rutinaria con resolutores como el CHAFF. Estas observaciones nos sugieren que alguna combinación de los comportamientos de la heurística de min-conflictos y de pasada-aleatoria nos proporciona una gran capacidad de *propósito-general* para resolver muchas situaciones en las que se requiere el razonamiento combinatorio.

7.7 Agentes basados en lógica proposicional

En esta sección, vamos a juntar lo que hemos aprendido hasta ahora para construir agentes que funcionan utilizando la lógica proposicional. Veremos dos tipos de agentes: aquellos que utilizan algoritmos de inferencia y una base de conocimiento, como el agente basado en conocimiento genérico de la Figura 7.1, y aquellos que evalúan directamente expresiones lógicas en forma de circuitos. Aplicaremos ambos tipos de agentes en el mundo de *wumpus*, y encontraremos que ambos sufren de serias desventajas.

Encontrar hoyos y *wumpus* utilizando la inferencia lógica

Permítanos empezar con un agente que razona mediante la lógica acerca de las localizaciones de los hoyos, de los *wumpus* y de la seguridad de las casillas. El agente comienza con una base de conocimiento que describe la «física» del mundo de *wumpus*. El agente sabe que la casilla [1, 1] no tiene ningún hoyo ni ningún *wumpus*; es decir, $\neg H_{1,1}$ y $\neg W_{1,1}$. El agente también conoce una sentencia que indica cómo se percibe una brisa en una casilla [x, y]:

$$B_{x,y} \Leftrightarrow (H_{x,\,y+1} \vee H_{x,\,y-1} \vee H_{x+1,\,y} \vee H_{x-1,\,y}) \tag{7.1}$$

El agente conoce una sentencia que indica cómo se percibe el hedor en una casilla [x, y]:

$$M_{x,y} \Leftrightarrow (W_{x,\,y+1} \vee W_{x,\,y-1} \vee W_{x+1,\,y} \vee W_{x-1,\,y}) \tag{7.2}$$

Finalmente, el agente sabe que sólo hay un *wumpus*. Esto se expresa de dos maneras. En la primera, debemos definir que hay *por lo menos* un *wumpus*:

$$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,3} \vee W_{4.4}$$

Entonces, debemos definir que *como mucho* hay un *wumpus*. Una manera de definirlo es diciendo que para dos casillas cualesquiera, una de ellas debe estar libre de un *wum-*

pus. Con n casillas, tenemos $n(n-1)/2$ sentencias del tipo $\neg W_{1,1} \vee \neg W_{1,2}$. Entonces, para un mundo de 4×4, comenzamos con un total de 155 sentencias conteniendo 64 símbolos diferentes.

El programa del agente, que se muestra en la Figura 7.19 DICE a su base de conocimiento cualquier nueva percepción acerca de una brisa o un mal hedor. (También actualiza algunas variables del programa para guardar la pista de dónde se encuentra y que casillas ha visitado. Estos últimos datos los necesitará más adelante.) Entonces el programa escoge dónde observar antes de entre las casillas que rodean al agente, es decir, las casillas adyacentes a aquellas ya visitadas. Una casilla $[i, j]$ que rodea al agente es *probable que sea segura* si la sentencia $(\neg H_{i,j} \wedge \neg W_{i,j})$ se deduce de la base de conocimiento. La siguiente alternativa mejor es una casilla *posiblemente segura*, o sea, aquella casilla para la cual el agente no puede demostrar si hay un hoyo o un *wumpus*, es decir, aquella para la que *no* se deduce $(H_{i,j} \vee W_{i,j})$.

El cálculo para averiguar la implicación mediante PREGUNTAR se puede implementar utilizando cualquiera de los métodos descritos anteriormente en este capítulo. ¿IMPLICACIÓN-TV? (Figura 7.10) es obviamente impracticable, ya que debería enumerar 2^{64} filas. El DPLL (Figura 7.16) realiza las inferencias necesarias en pocos milisegundos, principalmente gracias a la heurística de propagación unitaria. El SAT-CAMINAR también se puede utilizar, teniendo en cuenta la advertencia acerca de la incompletitud. En el mundo de *wumpus*, los obstáculos para encontrar un modelo, realizados 10.000 intercambios,

```
función AGENTE-WUMPUS-LP(percepción) devuelve una acción
  entradas: percepción, una lista, [hedor, brisa, resplandor]
  variables estáticas: BC, una base de conocimiento, inicialmente conteniendo la «física» del
                       mundo de wumpus
                       x, y, orientación, la posición del agente (inicialmente en 1, 1) y su
                       orientación (inicialmente derecha)
                       visitada, una matriz indicando qué casillas han sido visitadas,
                       inicialmente falso
                       acción, la acción más reciente del agente, inicialmente null

  plan, una secuencia de acciones, inicialmente vacía
  actualiza x, y, orientación, visitada basada en acción
  si hedor entonces DECIR(BC, M_{x,y}) sino DECIR(BC, ¬ M_{x,y})
  si brisa entonces DECIR(BC, B_{x,y}) sino DECIR(BC, ¬ B_{x,y})
  si resplandor entonces acción ← agarrar
  sino si plan no está vacío entonces acción ← POP(plan)
  sino si para alguna casilla [i, j] que nos rodea es verdadero PREGUNTAR(BC, (¬ H_{i,j} ∧ ¬ W_{i,j}))
          o es falso PREGUNTAR(BC, (H_{i,j} ∨ W_{i,j})) entonces hacer
     plan ← BUSQUEDA-GRAFO-A*(PROBLEMA-RUTA([x, y], orientación, [i, j], visitada))
     acción ← POP(plan)
  sino acción ← un movimiento escogido al azar
  devolver acción
```

Figura 7.19 Un agente en el mundo de wumpus que utiliza la lógica proposicional para identificar hoyos, wumpus y casillas seguras. La subrutina PROBLEMA-RUTA construye un problema de búsqueda cuya solución es una secuencia de acciones que nos llevan de $[x, y]$ a $[i, j]$ pasando sólo a través de las casillas previamente visitadas.

se corresponden invariablemente con la *insatisfacibilidad*, así que los errores no se deben probablemente a la incompletitud.

El programa AGENTE-WUMPUS-LP se comporta bastante bien en un mundo de *wumpus* pequeño. Sin embargo, sucede algo profundamente insatisfactorio respecto a la base de conocimiento del agente. La *BC* contiene las sentencias que definen la «física» en la forma dada en las Ecuaciones (7.1) y (7.2) *para cada casilla individual*. Cuanto más grande sea el entorno, más grande necesita ser la base de conocimiento inicial. Preferiríamos en mayor medida disponer sólo de las dos sentencias para definir cómo se presenta una brisa o un hedor en *cualquier* casilla. Esto está más allá del poder de expresión de la lógica proposicional. En el próximo capítulo veremos un lenguaje lógico más expresivo mediante el cual es más fácil expresar este tipo de sentencias.

Guardar la pista acerca de la localización y la orientación del agente

El programa del agente de la Figura 7.19 «hace trampa» porque guarda la pista de su localización *fuera* de la base de conocimiento, en vez de utilizar el razonamiento lógico[13]. Para hacerlo «correctamente» necesitaremos proposiciones acerca de la localización. Una primera aproximación podría consistir en utilizar un símbolo como $L_{1,1}$ para indicar que el agente se encuentra en la casilla [1, 1]. Entonces la base de conocimiento inicial podría incluir sentencias como

$$L_{1,1} \wedge OrientadoDerecha \wedge Avanzar \Rightarrow L_{2,1}$$

Vemos inmediatamente que esto no funciona correctamente. Si el agente comienza en la casilla [1, 1] orientado a la derecha y avanza, de la base de conocimiento se deduciría $L_{1,1}$ (la localización original del agente) y $L_{2,1}$ (su nueva localización). ¡Aunque estas dos proposiciones no pueden ser verdaderas a la vez! El problema es que las proposiciones de localización deberían referirse a dos instantes de tiempo diferentes. Necesitamos $L^1_{1,1}$ para indicar que el agente se encuentra en la casilla [1, 1] en el instante de tiempo 1, $L^2_{2,1}$ para indicar que el agente se encuentra en la casilla [2, 1] en el instante de tiempo 2, etcétera. Las proposiciones acerca de la orientación y la acción también necesitan depender del tiempo. Por lo tanto, la sentencia correcta es

$$L^1_{1,1} \wedge OrientadoDerecha^1 \wedge Avanzar^1 \Rightarrow L^2_{2,1}$$
$$OrientadoDerecha \wedge GirarIzquierda^1 \Rightarrow OrientadoArriba^2$$

De esta manera resulta bastante difícil construir una base de conocimiento completa y correcta para guardar la pista de cada cosa que sucede en el mundo de *wumpus*; pospondremos la discusión en su detalle para el Capítulo 10. Lo que nos proponemos hacer aquí es que la base de conocimiento inicial contenga sentencias como los dos ejemplos anteriores para cada instante de tiempo *t*, así como para cada localización. Es

[13] El lector que sea observador se habrá dado cuenta de que esto nos posibilitó afinar la conexión que hay entre las percepciones, como *Brisa* y la proposiciones acerca de la localización específica, como $B_{1,1}$.

decir, que para cada instante de tiempo t y localización $[x, y]$, la base de conocimiento contenga una sentencia del tipo

$$L^t_{x,y} \wedge OrientadoDerecha^t \wedge Avanzar^t \Rightarrow L^{t+1}_{x+1,y} \tag{7.3}$$

Aunque pongamos un límite superior de pasos permitidos en el tiempo (por ejemplo 100) acabamos con decenas de miles de sentencias. Se presenta el mismo problema si añadimos las sentencias «a medida que las necesitemos» en cada paso en el tiempo. Esta proliferación de cláusulas hace que la base de conocimiento sea ilegible para las personas, sin embargo, los resolutores rápidos en lógica proposicional aún pueden manejar un mundo de *wumpus* de 4×4 con cierta facilidad (éstos encuentran su límite en los mundos de 100×100 casillas). Los agentes basados en circuitos de la siguiente sección ofrecen una solución parcial a este problema de proliferación de las cláusulas, pero la solución íntegra deberá esperar hasta que hayamos desarrollado la lógica de primer orden en el Capítulo 8.

Agentes basados en circuitos

AGENTE BASADO EN CIRCUITOS

CIRCUITO SECUENCIAL

PUERTAS

REGISTROS

Un **agente basado en circuitos** es un tipo particular de agente reflexivo con estado interno, tal como se definió en el Capítulo 2. Las percepciones son las entradas de un **circuito secuencial** (una red de **puertas**, cada una de ellas implementa una conectiva lógica, y de **registros**, cada uno de ellos almacena el valor lógico de una proposición atómica). Las salidas del circuito son los registros que se corresponden con las acciones, por ejemplo, la salida *Agarrar* se asigna a *verdadero* si el agente quiere coger algo. Si la entrada *Resplandor* se conecta directamente a la salida *Agarrar*, el agente cogerá el objetivo (el objeto *oro*) siempre que vea el resplandor. (*Véase* Figura 7.20.)

Figura 7.20 Parte de un agente basado en circuitos para el mundo de *wumpus*, mostrando las entradas, las salidas, el circuito para coger el oro, y el circuito que determina si el *wumpus* está vivo. Los registros se muestran como rectángulos, y los retardos de un paso se muestran como pequeños triángulos.

FLUJO DE DATOS

Los circuitos se evalúan de igual modo que los **flujos de datos**: en cada instante de tiempo, se asignan las entradas y se propagan las señales a través del circuito. Siempre que una puerta disponga de todas sus entradas, ésta produce una salida. Este proceso está íntimamente relacionado con el de encadenamiento hacia delante en un grafo Y-O, como el de la Figura 7.15(b).

En la sección precedente dijimos que los agentes basados en circuitos manejan el tiempo de forma más satisfactoria que los agentes basados en la inferencia proposicional. Esto se debe a que cada registro nos da el valor de verdad de su correspondiente símbolo proposicional *en el instante de tiempo actual t*, en vez de disponer de una copia diferente para cada instante de tiempo. Por ejemplo, podríamos tener un registro para *Vivo* que debería contener el valor *verdadero* cuando el *wumpus* esté vivo, y *falso* cuando esté muerto. Este registro se corresponde con el símbolo proposicional $Vivo^t$, de esta manera, en cada instante de tiempo el registro se refiere a una proposición diferente. El estado interno del agente, es decir, su memoria, mantiene conectando hacia atrás la salida de un registro con el circuito mediante una **línea de retardo**. Este mecanismo nos da el valor del registro en el instante de tiempo previo. En la Figura 7.20 se muestra un ejemplo. El valor del registro *Vivo* se obtiene de la conjunción de la negación del registro *Grito* y de su propio valor anterior. En términos de proposiciones, el circuito del registro *Vivo* implementa la siguiente conectiva bicondicional

LÍNEA DE RETARDO

$$Vivo^t \Leftrightarrow \neg Grito^t \wedge Vivo^{t-1} \tag{7.4}$$

que nos dice que el *wumpus* está vivo en el instante *t si y sólo si* no se percibe ningún grito en el instante *t* y estaba vivo en el instante $t - 1$. Asumimos que el circuito se inicializa con el registro *Vivo* asignado a *verdadero*. Por lo tanto, *Vivo* permanecerá siendo verdadero hasta que haya un grito, con lo que se convertirá y se mantendrá en el valor falso. Esto es exactamente lo que deseamos.

La localización del agente se puede tratar, en mucho, de la misma forma que la salud del *wumpus*. Necesitamos un registro $L_{x,y}$ para cada *x* e *y*; su valor debería ser *verdadero* si el agente se encuentra en la casilla [*x, y*]. Sin embargo, el circuito que asigna el valor de $L_{x,y}$ es mucho más complicado que el circuito para el registro *Vivo*. Por ejemplo, el agente está en la casilla [1, 1] en el instante *t* si: (a) estaba allí en el instante $t - 1$ y no se movió hacia delante o lo intentó pero tropezó con un muro; o (b) estaba en la casilla [1, 2] orientado hacia abajo y avanzó; o (c) estaba en la casilla [2, 1] orientado a la izquierda y avanzó:

$$\begin{aligned} L^t_{1,1} \quad \Leftrightarrow \quad & (L^{t-1}_{1,1} \wedge (\neg Avanzar^{t-1} \vee Tropezar^t)) \\ & \vee (L^{t-1}_{1,2} \wedge (OrientadoAbajo^{t-1} \wedge Avanzar^{t-1})) \\ & \vee (L^{t-1}_{2,1} \wedge (OrientadoIzquierda^{t-1} \wedge Avanzar^{t-1})) \end{aligned} \tag{7.5}$$

En la Figura 7.21 se muestra el circuito para $L_{1,1}$. Cada registro de localización tiene enlazado un circuito similar a éste. En el Ejercicio 7.13(b) se pide diseñar un circuito para las proposiciones de orientación.

Los circuitos de las Figuras 7.20 y 7.21 mantienen los valores correctos de los registros *Vivo* y $L_{x,y}$ en todo momento. Sin embargo, estas proposiciones son extrañas, en el sentido de que *sus valores de verdad correctos siempre se pueden averiguar*. En lu-

Figura 7.21 El circuito para determinar si el agente está en la casilla [1, 1]. Cada registro de localización y de orientación tiene enlazado un circuito similar.

gar de ello, consideremos la proposición $B_{4,4}$: en la casilla [4, 4] se percibe una brisa. Aunque el valor de verdad de esta proposición se mantiene fijo, el agente no puede aprender su valor hasta que haya visitado la casilla [4, 4] (o haya deducido que hay un hoyo cercano). Las lógicas proposicional y de primer orden están diseñadas para representar proposiciones verdaderas, falsas, o inciertas, de forma automática. Los circuitos no pueden hacerlo: el registro $B_{4,4}$ debe contener *algún* valor, bien *verdadero* o *falso*, aun antes de que se descubra su valor de verdad. El valor del registro bien podría ser el erróneo, y esto nos llevaría a que el agente se extraviara. En otras palabras, necesitamos representar tres posibles estados (se sabe que $B_{4,4}$ es verdadero, falso, o desconocido) y sólo tenemos un *bit* para representar estos estados.

La solución a este problema es utilizar dos bits en vez de uno. $B_{4,4}$ se puede representar mediante dos registros, a los que llamaremos $K(B_{4,4})$ y $K(\neg B_{4,4})$, en donde K significa «conocido». (¡Tenga en cuenta que esta representación consiste tan sólo en unos símbolos con nombres complicados, aunque parezcan expresiones estructuradas!) Cuando ambas, $K(B_{4,4})$ y $K(\neg B_{4,4})$, son falsas, significa que el valor de verdad de $B_{4,4}$ es desconocido. (¡Si ambas son ciertas, entonces la base de conocimiento tiene un fallo!) A partir de ahora, utilizaríamos $K(B_{4,4})$ en vez de $B_{4,4}$, siempre que ésta aparezca en alguna parte del circuito. Por lo general, representamos cada proposición que es potencialmente indeterminada mediante dos **proposiciones acerca del conocimiento** para especificar si la proposición subyacente se sabe que es verdadera y se sabe que es falsa.

PROPOSICIONES ACERCA DEL CONOCIMIENTO

En breve veremos un ejemplo de cómo utilizar las proposiciones acerca del conocimiento. Primero necesitamos resolver cómo determinar los valores de verdad de las propias proposiciones acerca del conocimiento. Fíjese en que, mientras $B_{4,4}$ tiene un valor de verdad fijo, $K(B_{4,4})$ y $K(\neg B_{4,4})$ *cambian* a medida que el agente descubre más cosas acerca del mundo. Por ejemplo, $K(B_{4,4})$ comienza siendo falsa y entonces se transforma en verdadera tan pronto como se puede determinar $B_{4,4}$ que es verdadera (es decir, cuan-

do el agente está en la casilla [4, 4] y detecta una brisa). A partir de entonces permanece siendo verdadera. Así tenemos

$$K(B_{4,4})^t \Leftrightarrow K(B_{4,4})^{t-1} \vee (L^t_{4,4} \wedge Brisa^t) \tag{7.6}$$

Se puede escribir una ecuación similar para $K(\neg B_{4,4})^t$.

Ahora que el agente sabe acerca de las casillas con brisa, puede ocuparse de los hoyos. La ausencia de un hoyo en una casilla se puede averiguar si y sólo si se sabe que en una de sus casillas vecinas no hay ninguna brisa. Por ejemplo, tenemos

$$K(\neg H_{4,4})^t \Leftrightarrow K(\neg B_{3,4})^t \vee K(\neg B_{4,3})^t \tag{7.7}$$

Determinar si *hay* un hoyo en una casilla es más difícil, debe haber una brisa en una casilla adyacente que no sea causada por otro hoyo:

$$\begin{array}{lcl} K(H_{4,4})^t & \Leftrightarrow & (K(B_{3,4})^t \wedge K(\neg H_{2,4})^t \wedge K(\neg H_{3,3})^t) \\ & \vee & (K(B_{4,3})^t \wedge K(\neg H_{4,2})^t \wedge K(\neg H_{3,3})^t) \end{array} \tag{7.8}$$

Mientras que utilizar los circuitos para determinar la presencia o ausencia de hoyos puede resultar algo peliagudo, *sólo se necesitan un número constante de puertas para cada casilla*. Esta propiedad es esencial si debemos construir agentes basados en circuitos que se pueden ampliar de forma razonable. En realidad ésta es una propiedad del propio mundo de *wumpus*; decimos que un entorno manifiesta **localidad** si el valor de verdad de una proposición relevante se puede obtener observando sólo un número constante de otras proposiciones. La localidad es muy sensible a la «física» precisa del entorno. Por ejemplo, el dominio de los dragaminas (Ejercicio 7.11) no es un dominio localista, porque determinar si una mina se encuentra en una casilla dada puede requerir observar las casillas que están arbitrariamente lejos. Los agentes basados en circuitos no son siempre practicables para los dominios no localistas.

ACÍCLICO

Hay un asunto por el que hemos caminado de puntillas y con mucho cuidado: el tema es la propiedad de ser **acíclico**. Un circuito es acíclico si cada uno de sus caminos en el que se conecta hacia atrás la salida de un registro con su entrada se realiza mediante un elemento de retardo. ¡Necesitamos que todos los circuitos sean acíclicos porque los que no lo son, al igual que los artefactos físicos, no funcionan! Éstos pueden entrar en oscilaciones inestables produciendo valores indefinidos. Como ejemplo de un circuito cíclico, tenga en cuenta la siguiente ampliación de la Ecuación (7.6):

$$K(B_{4,4})^t \Leftrightarrow K(B_{4,4})^{t-1} \vee (L^t_{4,4} \wedge Brisa^t) \vee K(H_{3,4})^t \vee K(H_{4,3})^t \tag{7.9}$$

Los disyuntores extras, $K(H_{3,4})^t$ y $K(H_{4,3})^t$, permiten al agente determinar si hay brisa a partir del conocimiento de la presencia de hoyos en las casillas adyacentes, algo que parece totalmente razonable. Pero ahora, desafortunadamente, tenemos que la detección de la brisa depende de los hoyos adyacentes, y la detección de los hoyos depende de las brisas adyacentes, si tenemos en cuenta la Ecuación (7.8). Por lo tanto el circuito, en su conjunto, tendría ciclos.

La dificultad no es que la Ecuación (7.9) ampliada sea *incorrecta*. Más bien, el problema consiste en que las dependencias entrelazadas que se presentan en estas ecuaciones no se pueden resolver por el simple mecanismo de la propagación de los valores de

verdad en el correspondiente circuito Booleano. La versión acíclica utilizando la Ecuación (7.6), que determina si hay brisa sólo a partir de las observaciones directas en la casilla es *incompleta*, en el sentido de que en algunos puntos, el agentes basado en circuitos podría saber menos que un agente basado en inferencia utilizando un procedimiento de inferencia completo. Por ejemplo, si hay una brisa en la casilla [1, 1], el agente basado en inferencia puede concluir que también hay una brisa en la casilla [2, 2], mientras que el agente basado en circuitos no puede hacerlo, utilizando la Ecuación (7.6). *Se puede* construir un circuito completo (después de todo, los circuitos secuenciales pueden emular cualquier computador digital) pero sería algo significativamente más complejo.

Una comparación

Los agentes basados en inferencia y los basados en circuitos representan los extremos declarativo y procesal en el diseño de agentes. Se pueden comparar según diversas dimensiones:

- *Precisión*: el agente basado en circuitos, a diferencia del agente basado en inferencia, no necesita disponer de copias diferentes de su «conocimiento» para cada instante de tiempo. En vez de ello, éste sólo se refiere al instante actual y al previo. Ambos agentes necesitan copias de la «física» de cada casilla (expresada mediante sentencias o circuitos), y por lo tanto, no se amplían adecuadamente a entornos mayores. En entornos con muchos objetos relacionados de forma compleja el número de proposiciones abrumará a cualquier agente proposicional. Este tipo de entornos requiere del poder expresivo de la lógica de primer orden. (*Véase* Capítulo 8.) Además, ambos tipos de agente proposicional están poco preparados para expresar o resolver el problema de encontrar un camino a una casilla segura que esté cercana. (Por esta razón, el AGENTE-WUMPUS-LP recurre a un algoritmo de búsqueda.)
- *Eficiencia computacional*: en el *peor de los casos*, la inferencia puede tomar un tiempo exponencial respecto al número de símbolos, mientras que evaluar un circuito toma un tiempo lineal respecto al tamaño del circuito (o lineal respecto a la *intensidad de integración* del circuito, si éste se construye como un dispositivo físico). Sin embargo, vemos que en la *práctica*, el DPLL realiza las inferencias requeridas bastante rápidamente[14].
- *Completitud*: anteriormente insinuamos que el agente basado en circuitos podría ser incompleto, debido a la restricción de que sea acíclico. Las causas de la incompletitud son en realidad más básicas. Primero, recordemos que un circuito se ejecuta en tiempo lineal respecto a su tamaño. Esto significa que, para algunos entornos, un circuito que sea completo (a saber, uno que calcula el valor de verdad de cada proposición determinable) debe ser exponencialmente más grande que la *BC* de un agente basado en inferencia. Dicho de otro modo, deberíamos poder resolver el problema de la implicación proposicional en menor tiempo que el tiempo exponencial, lo que parece improbable. Una segunda causa es la naturaleza del

[14] De hecho, ¡todas las inferencias hechas por un circuito se pueden hacer por el DPLL en tiempo lineal! Esto es debido a que la evaluación de un circuito, al igual que el encadenamiento hacia delante, se puede emular mediante el DPLL, utilizando la regla de propagación unitaria.

estado interno del agente. El agente basado en inferencia recuerda cada percepción que ha tenido, y conoce, bien implícita o explícitamente, cada sentencia que se sigue de las percepciones y la *BC* inicial. Por ejemplo, dado $B_{1,1}$, el agente conoce la disyunción $H_{1,2} \vee H_{2,1}$, de lo cual se sigue $B_{2,2}$. Por el otro lado, el agente basado en circuitos olvida todas sus percepciones anteriores y recuerda tan sólo las proposiciones individuales almacenadas en los registros. De este modo, $H_{1,2}$ y $H_{2,1}$ permanecen desconocidas *cada una de ellas* aún después de la primera percepción, así que no se sacará ninguna conclusión acerca de $B_{2,2}$.

- *Facilidad de construcción*: este es un aspecto muy importante acerca del cual es difícil ser precisos. Desde luego, los autores encontramos mucho más fácil especificar la «física» de forma declarativa, mientras que idear pequeños circuitos acíclicos, y no demasiado incompletos, para la detección directa de hoyos, nos ha parecido bastante dificultoso.

En suma, parece que hay *compensaciones* entre la eficiencia computacional, la concisión, la completitud y la facilidad de construcción. Cuando la conexión entre las percepciones y las acciones es simple (como en la conexión entre *Resplandor* y *Agarrar*) un circuito parece que es óptimo. En un dominio como el ajedrez, por ejemplo, las reglas declarativas son concisas y fácilmente codificables (como mínimo en la lógica de primer orden), y en cambio, utilizar un circuito para calcular los movimientos directamente a partir de los estados del tablero, sería un esfuerzo inimaginablemente enorme.

Podemos observar estos diferentes tipos de compensaciones en el reino animal. Los animales inferiores, con sus sistemas nerviosos muy sencillos, quizá estén basados en circuitos, mientras que los animales superiores, incluyendo a los humanos, parecen realizar inferencia con base en representaciones explícitas. Esta característica les permite ejecutar funciones mucho más complejas de un agente. Los humanos también disponen de circuitos para implementar sus reflejos, y quizá, también para **compilar** sus representaciones declarativas en circuitos, cuando ciertas inferencias pasan a ser una rutina. En este sentido, el diseño de un **agente híbrido** (*véase* Capítulo 2) podría poseer lo mejor de ambos mundos.

COMPILACIÓN

7.8 Resumen

Hemos introducido los agentes basados en conocimiento y hemos mostrado cómo definir una lógica con la que los agentes pueden razonar acerca del mundo. Los principales puntos son los siguientes:

- Los agentes inteligentes necesitan el conocimiento acerca del mundo para tomar las decisiones acertadas.
- Los agentes contienen el conocimiento en forma de **sentencias** mediante un **lenguaje de representación del conocimiento**, las cuales quedan almacenadas en una **base de conocimiento**.
- Un agente basado en conocimiento se compone de una base de conocimiento y un mecanismo de inferencia. El agente opera almacenando las sentencias acerca del mundo en su base de conocimiento, utilizando el mecanismo de inferencia para

inferir sentencias nuevas, y utilizando estas sentencias nuevas para decidir qué acción debe tomar.

- Un lenguaje de representación del conocimiento se define por su **sintaxis**, que especifica la estructura de las sentencias, y su **semántica**, que define el **valor de verdad** de cada sentencia en cada **mundo posible**, o **modelo**.
- La relación de **implicación** entre las sentencias es crucial para nuestro entendimiento acerca del razonamiento. Una sentencia α implica otra sentencia β si β es verdadera en todos los mundos donde α lo es. Las definiciones familiares a este concepto son: la **validez** de la sentencia $\alpha \Rightarrow \beta$, y la ***insatisfacibilidad*** de la sentencia $\alpha \wedge \neg\beta$.
- La inferencia es el proceso que consiste en derivar nuevas sentencias a partir de las ya existentes. Los algoritmos de inferencia **sólidos** *sólo* derivan aquellas sentencias que son implicadas; los algoritmos **completos** derivan todas las sentencias implicadas.
- La **lógica proposicional** es un lenguaje muy sencillo compuesto por los **símbolos proposicionales** y las **conectivas lógicas**. De esta manera se pueden manejar proposiciones que se sabe son ciertas, falsas, o completamente desconocidas.
- El conjunto de modelos posibles, dado un vocabulario proposicional fijado, es finito, y así se puede comprobar la implicación tan sólo enumerando los modelos. Los algoritmos de inferencia basados en la **comprobación de modelos** más eficientes para la lógica proposicional, entre los que se encuentran los métodos de búsqueda local y *backtracking*, a menudo pueden resolver problemas complejos muy rápidamente.
- Las **reglas de inferencia** son patrones de inferencia sólidos que se pueden utilizar para encontrar demostraciones. De la regla de **resolución** obtenemos un algoritmo de inferencia completo para bases de conocimiento que están expresadas en **forma normal conjuntiva**. El **encadenamiento hacia delante** y el **encadenamiento hacia atrás** son algoritmos de razonamiento muy adecuados para bases de conocimiento expresadas en **cláusulas de Horn**.
- Se pueden diseñar dos tipos de agentes que utilizan la lógica proposicional: los **agentes basados en inferencia** utilizan algoritmos de inferencia para guardar la pista del mundo y deducir propiedades ocultas, mientras que los **agentes basados en circuitos** representan proposiciones mediante bits en registros, y los actualizan utilizando la propagación de señal de los circuitos lógicos.
- La lógica proposicional es razonablemente efectiva para ciertas tareas de un agente, pero no se puede escalar para entornos de tamaño ilimitado, a causa de su falta de poder expresivo para manejar el tiempo de forma precisa, el espacio, o patrones genéricos de relaciones entre objetos.

Notas bibliográficas e históricas

La ponencia «Programas con Sentido Común» de John McCarthy (McCarthy, 1958, 1968) promulgaba la noción de agentes que utilizan el razonamiento lógico para mediar entre sus percepciones y sus acciones. En él también erigió la bandera del declarativis-

mo, señalando que decirle a un agente lo que necesita saber es una forma muy elegante de construir *software*. El artículo «El Nivel de Conocimiento» de Allen Newell (1982) propone que los agentes racionales se pueden describir y analizar a un nivel abstracto definido a partir del conocimiento que el agente posee más que a partir de los programas que ejecuta. En Boden (1977) se comparan los enfoques declarativo y procedural en la IA. Este debate fue reanimado, entre otros, por Brooks (1991) y Nilsson (1991).

La Lógica tiene sus orígenes en la Filosofía y las Matemáticas de los Griegos antiguos. En los trabajos de Platón se encuentran esparcidos diversos principios lógicos (principios que conectan la estructura sintáctica de las sentencias con su verdad o falsedad, con su significado, o con la validez de los argumentos en los que aparecen). El primer estudio sistemático acerca de la lógica que se conoce lo llevó a cabo Aristóteles, cuyo trabajo fue recopilado por sus estudiantes después de su muerte, en el 322 a.C., en un tratado denominado *Organon*. Los **silogismos** de Aristóteles fueron lo que ahora podríamos llamar reglas de inferencia. Aunque los silogismos tenían elementos, tanto de la lógica proposicional como de la de primer orden, el sistema como un todo era algo endeble según los estándares actuales. No permitió aplicar los patrones de inferencia a sentencias de complejidad diversa, cosa que sí lo permite la lógica proposicional moderna.

SILOGISMOS

Las escuelas íntimamente ligadas de los estoicos y los megarianos (que se originaron en el siglo V d.C. y continuaron durante varios siglos después) introdujeron el estudio sistemático de la implicación y otras estructuras básicas que todavía se utilizan en la lógica proposicional moderna. La utilización de las tablas de verdad para definir las conectivas lógicas se debe a Filo de Megara. Los estoicos tomaron como válidas cinco reglas de inferencia básicas sin demostrarlas, incluyendo la regla que ahora denominamos Modus Ponens. Derivaron un buen número de reglas a partir de estas cinco, utilizando entre otros principios, el teorema de la deducción (página 236) y fueron mucho más claros que Aristóteles acerca del concepto de demostración. Los estoicos afirmaban que su lógica era completa, en el sentido de ser capaz de reproducir todas las inferencias válidas; sin embargo, lo que de ellos queda es un conjunto de explicaciones demasiado fragmentadas. Que se sepa, un buen relato acerca de las lógicas Megariana y Estoica es el texto de Benson Mates (1953).

La idea de reducir la lógica a un proceso puramente mecánico, aplicado a un lenguaje formal es de Leibniz (1646-1716). Sin embargo, su propia lógica matemática era gravemente deficiente, y él es mucho más recordado simplemente por introducir estas ideas como objetivos a alcanzar que por sus intentos de lograrlo.

George Boole (1847) introdujo el primer sistema detallado y factible sobre lógica formal en su libro *El análisis Matemático de la Lógica*. La lógica de Boole estaba totalmente modelada a partir del álgebra de los números reales y utilizó la sustitución de expresiones lógicamente equivalentes como su principal método de inferencia. Aunque el sistema de Boole no abarcaba toda la lógica proposicional, estaba lo bastante cerca como para que otros matemáticos completaran las lagunas. Schröder (1877) describió la forma normal conjuntiva, mientras que las cláusulas de Horn fueron introducidas mucho más tarde por Alfred Horn (1951). La primera explicación completa acerca de la lógica proposicional moderna (y de la lógica de primer orden) se encuentra en el *Begriffschrift* («Escritura de Conceptos» o «Notación conceptual») de Gottlob Frege (1879).

El primer artefacto mecánico para llevar a cabo inferencias lógicas fue construido por el tercer Conde de Stanhope (1753-1816). El demostrador de Stanhope podía manejar silogismos y ciertas inferencias con la teoría de las probabilidades. William Stanley Jevons, uno de esos que hizo mejoras y amplió el trabajo de Boole, construyó su «piano lógico» en 1869, artefacto para realizar inferencias en lógica Booleana. En el texto de Martín Gardner (1968) se encuentra una historia entretenida e instructiva acerca de estos y otros artefactos mecánicos modernos utilizados para el razonamiento. El primer programa de computador para la inferencia lógica publicado fue el Teórico Lógico de Newell, Shaw y Simon (1957). Este programa tenía la intención de modelar los procesos del pensamiento humano. De hecho, Martin Davis (1957) había diseñado un programa similar que había presentado en una demostración en 1954, pero los resultados del Teórico Lógico fueron publicados un poco antes. Tanto el programa de 1954 de Davis como el Teórico Lógico estaban basados en algunos métodos *ad hoc* que no influyeron fuertemente más tarde en la deducción automática.

Las tablas de verdad, como un método para probar la validez o *insatisfacibilidad* de las sentencias en el lenguaje de la lógica proposicional, fueron introducidas por separado, por Ludwig Wittgenstein (1922) y Emil Post (1921). En los años 30 se realizaron una gran cantidad de avances en los métodos de inferencia para la lógica de primer orden. En concreto, Gödel (1930) demostró que se podía obtener un procedimiento completo para la inferencia en lógica de primer orden, mediante su reducción a la lógica proposicional utilizando el teorema de Herbrand (Herbrand, 1930). Retomaremos otra vez esta historia en el Capítulo 9; aquí el punto importante es que el desarrollo de los algoritmos proposicionales eficientes de los años 60 fue causado en gran parte, por el interés de los matemáticos en un demostrador de teoremas efectivo para la lógica de primer orden. El algoritmo de Davis y Putnam (Davis y Putnam, 1960) fue el primer algoritmo efectivo para la resolución proposicional, pero en muchos casos era menos eficiente que el algoritmo DPLL con *backtracking* introducido dos años después (1962). En un trabajo de gran influencia de J. A. Robinson (1965) apareció la regla de resolución general y una demostración de su completitud, en el que también se mostró cómo razonar con lógica de primer orden, sin tener que recurrir a las técnicas proposicionales.

Stephen Cook (1971) demostró que averiguar la *satisfacibilidad* de una sentencia en lógica proposicional es un problema NP-completo. Ya que averiguar la implicación es equivalente a averiguar la *insatisfacibilidad*, éste es un problema co-NP-completo. Se sabe que muchos subconjuntos de la lógica proposcional se pueden resolver en un tiempo polinómico; las cláusulas de Horn son uno de estos subconjuntos. El algoritmo de encadenamiento hacia delante en tiempo lineal para las cláusulas de Horn se debe a Dowling y Gallier (1984), quienes describen su algoritmo como un proceso de flujo de datos parecido a la propagación de señales en un circuito. La *satisfacibilidad* ha sido uno de los ejemplos canónicos para las reducciones NP; por ejemplo, Kaye (2000) demostró que el juego del dragaminas (*véase* Ejercicio 7.11) es NP-completo.

Los algoritmos de búsqueda local para la *satisfacibilidad* se intentaron por diversos autores en los 80; todos los algoritmos estaban basados en la idea de minimizar las cláusulas insatisfechas (Hansen y Jaumard, 1990). Un algoritmo particularmente efectivo fue desarrollado por Gu (1989) e independientemente por Selman *et al*. (1992), quien lo lla-

mó GSAT y demostró que era capaz de resolver muy rápidamente un amplio rango de problemas muy duros. El algoritmo SAT-CAMINAR descrito en el capítulo es de Selman *et al*. (1996).

La «transición de fase» en los problemas aleatorios *k*-SAT de *satisfacibilidad* fue identificada por primera vez por Simon y Dubois (1989). Los resultados empíricos de Crawford y Auton (1993) sugieren que se encuentra en el valor del ratio cláusula/variable alrededor de 4,24 para problemas grandes de 3-SAT; este trabajo también describe una implementación muy eficiente del DPLL. (Bayardo y Schrag, 1997) describen otra implementación eficiente del DPLL utilizando las técnicas de satisfacción de restricciones, y (Moskewicz *et al*. 2001) describen el CHAFF, que resuelve problemas de verificación de *hardware* con millones de variables y que fue el ganador de la Competición SAT 2002. Li y Anbulagan (1997) analizan las heurísticas basadas en la propagación unitaria que permiten que los resolutores sean más rápidos. Cheeseman *et al*. (1991) proporcionan datos acerca de un número de problemas de la misma familia y conjeturan que todos los problemas NP duros tienen una transición de fase. Kirkpatrick y Selman (1994) describen mecanismos mediante los cuales la física estadística podría aclarar de forma precisa las «condiciones» que definen la transición de fase. Los análisis teóricos acerca de su *localización* son todavía muy flojos: todo lo que se puede demostrar es que se encuentra en el rango [3.003, 4.598] para 3-SAT aleatorio. Cook y Mitchell (1997) hacen un excelente repaso de los resultados en este y otros temas relacionados con la *satisfacibilidad*.

Las investigaciones teóricas iniciales demostraron que DPLL tiene una complejidad media polinómica para ciertas ditribuciones normales de problemas. Este hecho, potencialmente apasionante, pasó a ser menos apasionante cuando Franco y Paull (1983) demostraron que los mismos problemas se podían resolver en tiempo constante simplemente utilizando asignaciones aleatorias. El método de generación aleatoria descrito en el capítulo tiene problemas más duros. Motivados por el éxito empírico de la búsqueda local en estos problemas, Koutsoupias y Papadimitriou (1992) demostraron que un algoritmo sencillo de ascensión de colina puede resolver muy rápido *casi todas* las instancias del problema de la *satisfacibilidad*; sugiriendo que los problemas duros son poco frecuentes. Más aún, Schöning (1999) presentó una variante aleatoria de GSAT cuyo tiempo de ejecución esperado en el *peor de los casos* era de 1.333^n para los problemas 3-SAT (todavía exponencial, pero sustancialmente más rápido que los límites previos para los peores casos). Los algoritmos de *satisfacibilidad* son todavía una área de investigación muy activa; la colección de artículos de Du *et al*. (1999) proporciona un buen punto de arranque.

Los agentes basados en circuitos se remontan al trabajo de gran influencia de McCulloch y Pitts (1943), quienes iniciaron el campo de las redes neuronales. Al contrario del supuesto popular, el trabajo trata de la implementación del diseño de un agente basado en circuitos Booleanos en su cerebro. Sin embargo, los agentes basados en circuitos han recibido muy poca atención en la IA. La excepción más notable es el trabajo de Stan Rosenschein (Rosenschein, 1985; Kaelbling y Rosenschein, 1990), quienes desarrollaron mecanismos para compilar agentes basados en circuitos a partir de las descripciones declarativas del entorno y la tarea. Los circuitos para actualizar las proposiciones almacenadas en registros están íntimamente relacionados con el **axioma del estado sucesor** desarrollado para la lógica de primer orden por Reiter (1991). El trabajo de Rod Brooks

(1986, 1989) demuestra la efectividad de los diseños basados en circuitos para controlar robots (un tema del que nos ocuparemos en el Capítulo 25). Brooks (1991) sostiene que los diseños basados en circuitos son *todo* lo que se necesita en la IA (dicha representación y razonamiento es algo incómodo, costoso, e innecesario). Desde nuestro punto de vista, ningún enfoque es suficiente por sí mismo.

El mundo de *wumpus* fue inventado por Gregory Yob (1975). Irónicamente, Yob lo desarrolló porque estaba aburrido de los juegos basados en una matriz: la topología de su mundo de *wumpus* original era un dodecahedro; nosotros lo hemos retornado a la aburrida matriz. Michael Genesereth fue el primero en sugerir que se utilizara el mundo de *wumpus* para evaluar un agente.

EJERCICIOS

7.1 Describa el mundo de *wumpus* según las características de entorno tarea listadas en el Capítulo 2.

7.2 Suponga que el agente ha avanzado hasta el instante que se muestra en la Figura 7.4(a), sin haber percibido nada en la casilla [1, 1], una brisa en la [2, 1], y un hedor en la [1, 2], y en estos momentos está interesado sobre las casillas [1, 3], [2, 2] y [3, 1]. Cada una de ellas puede tener un hoyo y como mucho en una se encuentra el *wumpus*. Siguiendo el ejemplo de la Figura 7.5, construya el conjunto de los mundos posibles. (Debería encontrar unos 32 mundos.) Marque los mundos en los que la BC es verdadera y aquellos en los que cada una de las siguientes sentencias es verdadera:

α_2 = «No hay ningún hoyo en la casilla [2, 2]»
α_3 = «Hay un *wumpus* en la casilla [1, 3]»

Por lo tanto, demuestre que $BC \models \alpha_2$ y $BC \models \alpha_3$.

7.3 Considere el problema de decidir si una sentencia en lógica proposicional es verdadera dado un modelo.

a) Escriba un algoritmo recursivo ¿VERDADERA-LP?(s, m) que devuelva *verdadero* si y sólo si la sentencia s es verdadera en el modelo m (donde el modelo m asigna un valor de verdad a cada símbolo de la sentencia s). El algoritmo debería ejecutarse en tiempo lineal respecto al tamaño de la sentencia. (De forma alternativa, utiliza una versión de esta función del repositorio de código en línea, *online*.)

b) Dé tres ejemplos de sentencias de las que se pueda determinar si son verdaderas o falsas dado un modelo parcial, que no especifique el valor de verdad de algunos de los símbolos.

c) Demuestre que el valor de verdad (si lo tiene) de una sentencia en un modelo parcial no se puede determinar, por lo general, eficientemente.

d) Modifique el algoritmo ¿VERDADERA-LP? para que algunas veces pueda juzgar la verdad a partir de modelos parciales, manteniendo su estructura recursiva y tiempo de ejecución lineal. Dé tres ejemplos de sentencias de las cuales *no* se detecte su verdad en un modelo parcial mediante su algoritmo.

e) Investigue si el algoritmo modificado puede hacer que ¿IMPLICACIÓN-TV? sea más eficiente.

7.4 Demuestre cada una de las siguientes aserciones:

a) α es válido si y sólo si *Verdadero* $\models \alpha$.
b) Para cualquier α, *Falso* $\models \alpha$.
c) $\alpha \models \beta$ si y sólo si la sentencia $(\alpha \Rightarrow \beta)$ es válida.
d) $\alpha \equiv \beta$ si y sólo si la sentencia $(\alpha \Leftrightarrow \beta)$ es válida.
e) $\alpha \models \beta$ si y sólo si la sentencia $(\alpha \wedge \neg\beta)$ es *insatisfacible*.

7.5 Considere un vocabulario con sólo cuatro proposiciones, A, B, C, y D. ¿Cuántos modelos hay para las siguientes sentencias?

a) $(\mathrm{A} \wedge \mathrm{B}) \vee (\mathrm{B} \wedge \mathrm{C})$
b) $\mathrm{A} \vee \mathrm{B}$
c) $\mathrm{A} \Leftrightarrow \mathrm{B} \Leftrightarrow \mathrm{C}$

7.6 Hemos definido cuatro conectivas lógicas binarias.

a) ¿Hay otras conectivas que podrían ser útiles?
b) ¿Cuántas conectivas binarias puede haber?
c) ¿Por qué algunas de ellas no son muy útiles?

7.7 Utilizando un método a tu elección, verifique cada una de las equivalencias de la Figura 7.11.

7.8 Demuestre para cada una de las siguientes sentencias, si es válida, *insatisfacible*, o ninguna de las dos cosas. Verifique su decisión utilizando las tablas de verdad o las equivalencias de la Figura 7.11.

a) $Humo \Rightarrow Humo$
b) $Humo \Rightarrow Fuego$
c) $(Humo \Rightarrow Fuego) \Rightarrow (\neg Humo \Rightarrow \neg Fuego)$
d) $Humo \vee Fuego \vee \neg Fuego$
e) $((Humo \wedge Calor) \Rightarrow Fuego) \Leftrightarrow ((Humo \Rightarrow Fuego) \vee (Calor \Rightarrow Fuego))$
f) $(Humo \Rightarrow Fuego) \Rightarrow ((Humo \wedge Calor) \Rightarrow Fuego)$
g) $Grande \vee Mudo \vee (Grande \Rightarrow Mudo)$
h) $(Grande \wedge Mudo) \vee \neg Mudo$

7.9 (Adaptado de Barwise y Etchemendy (1993).) Dado el siguiente párrafo, ¿puede demostrar que el unicornio es un animal mitológico? ¿que es mágico?, ¿que tiene cuernos?

> Si el unicornio es un animal mitológico, entonces es inmortal, pero si no es mitológico, entonces es un mamífero mortal. Si el unicornio es inmortal o mamífero, entonces tiene cuernos. El unicornio es mágico si tiene cuernos.

7.10 Cualquier sentencia en lógica proposicional es lógicamente equivalente a la aserción de que no se presenta el caso en que cada mundo posible la haga falsa. Demuestre a partir de esta observación que cualquier sentencia se puede escribir en FNC.

7.11 El muy conocido juego del dragaminas está íntimamente relacionado con el mundo de *wumpus*. Un mundo del dragaminas es una matriz rectangular de N casillas con M minas invisibles esparcidas por la matriz. Cualquier casilla puede ser visitada por el agente; si se encuentra que hay una mina obtiene una muerte instantánea. El dragaminas indica la presencia de minas mostrando en cada casilla visitada el *número* de minas que se encuentran alrededor directa o diagonalmente. El objetivo es conseguir visitar todas las casillas libres de minas.

a) Dejemos que $X_{i,j}$ sea verdarero si y sólo si la casilla $[i, j]$ contiene una mina. Anote en forma de sentencia la aserción de que hay exactamente dos minas adyacentes a la casilla [1, 1] apoyándose en alguna combinación lógica de proposiciones con $X_{i,j}$.

b) Generalice su aserción de la pregunta (a) explicando cómo construir una sentencia en FNC que aserte que k de n casillas vecinas contienen minas.

c) Explique de forma detallada, cómo un agente puede utilizar DPLL para demostrar que una casilla contiene (o no) una mina, ignorando la restricción global de que hay exactamente M minas en total.

d) Suponga que la restricción global se construye mediante su método de la pregunta (b). ¿Cómo depende el número de cláusulas de M y N? Proponga una variación del DPLL para que la restricción global no se tenga que representar explícitamente.

e) ¿Algunas conclusiones derivadas con el método de la pregunta (c) son invalidadas cuando se tiene en cuenta la restricción global?

f) Dé ejempos de configuraciones de explorar valores que induzcan *dependencias a largo plazo* como la que genera que el contenido de una casilla no explorada daría información acerca del contenido de una casilla bastante distante. [*Pista*: pruebe primero con un tablero de $N \times 1$.]

7.12 Este ejercicio trata de la relación entre las cláusulas y las sentencias de implicación.

a) Demuestre que la cláusula $(\neg P_1 \vee \cdots \vee \neg P_m \vee Q)$ es lógicamente equivalente a la implicación $(P_1 \wedge \cdots \wedge P_m) \Rightarrow Q$.

b) Demuestre que cada cláusula (sin tener en cuenta el número de literales positivos) se puede escribir en la forma $(P_1 \wedge \cdots \wedge P_m) \Rightarrow (Q_1 \vee \cdots \vee Q_n)$, donde las Ps y las Qs son símbolos proposicionales. Una base de conocimiento compuesta por este tipo de sentencia está en **forma normal implicativa** o **forma de Kowalski**.

c) Interprete la regla de resolución general para las sentencias en forma normal implicativa.

7.13 En este ejercicio diseñaremos más cosas del agente *wumpus* basado en circuitos.

a) Escriba una ecuación, similar a la Ecuación (7.4), para la proposición *Flecha*, que debería ser verdadera cuando el agente aún tiene una flecha. Dibuje su circuito correspondiente.

b) Repita la pregunta (a) para *OrientadoDerecha*, utilizando la Ecuación (7.5) como modelo.

c) Cree versiones de las Ecuaciones 7.7 y 7.8 para encontrar el *wumpus*, y dibuje el circuito.

7.14 Discuta lo que quiere significar un comportamiento *óptimo* en el mundo de *wumpus*. Demuestre que nuestra definición de AGENTE-WUMPUS-LP no es óptima, y sugiera mecanismos para mejorarla.

7.15 Amplíe el AGENTE-WUMPUS-LP para que pueda guardar la pista de todos los hechos relevantes que están *dentro* de su base de conocimiento.

7.16 ¿Cuánto se tarda en demostrar que $BC \models \alpha$ utilizando el DPLL cuando α es un literal que ya está en la BC? Explíquelo.

7.17 Traza el comportamiento del DPLL con la base de conocimiento de la Figura 7.15 cuando intenta demostrar Q, y compare este comportamiento con el del algoritmo del encadenamiento hacia delante.

8 Lógica de primer orden

Donde nos daremos cuenta de que el mundo está bendecido con muchos objetos, que algunos de los cuales están relacionados con otros objetos, y nos esforzamos en razonar sobre ellos.

En el Capítulo 7 demostramos cómo un agente basado en el conocimiento podía representar el mundo en el que actuaba y deducir qué acciones debía llevar a cabo. En el capítulo anterior utilizamos la lógica proposicional como nuestro lenguaje de representación, porque nos bastaba para ilustrar los conceptos fundamentales de la lógica y de los agentes basados en el conocimiento. Desafortunadamente, la lógica proposicional es un lenguaje demasiado endeble para representar de forma precisa el conocimiento de entornos complejos. En este capítulo examinaremos la **lógica de primer orden**[1] que es lo suficientemente expresiva como para representar buena parte de nuestro conocimiento de sentido común. La lógica de predicados también subsume, o forma la base para, muchos otros lenguajes de representación y ha sido estudiada intensamente durante varias décadas. En la Sección 8.1 comenzamos con una breve discusión general acerca de los lenguajes de representación; en la Sección 8.2, se muestra la sintaxis y la semántica de la lógica de primer orden; y en la Sección 8.3 se ilustra el uso de la lógica de primer orden en representaciones sencillas.

LÓGICA DE PRIMER ORDEN

8.1 Revisión de la representación

En esta sección, discutiremos acerca de la naturaleza de los lenguajes de representación. Esta discusión nos llevará al desarrollo de la lógica de primer orden, un lenguaje mu-

[1] También denominada **Cálculo de Predicados** (de **Primer Orden**), algunas veces se abrevia mediante LP o CP.

cho más expresivo que la lógica proposicional, que introdujimos en el Capítulo 7. Veremos la lógica proposicional y otros tipos de lenguajes para entender lo que funciona y lo que no. Nuestra discusión será rápida, resumiendo siglos del pensamiento humano, de ensayo y error, todo ello en unos pocos párrafos.

Los lenguajes de programación (como C++, o Java, o Lisp) son, de lejos, la clase más amplia de lenguajes formales de uso común. Los programas representan en sí mismos, y de forma directa, sólo procesos computacionales. Las estructuras de datos de los programas pueden representar hechos; por ejemplo, un programa puede utilizar una matriz de 4×4 para representar el contenido del mundo de *wumpus*. De esta manera, una sentencia de un lenguaje de programación como *Mundo*[2, 2] ← *Hoyo*, es una forma bastante natural de expresar que hay un hoyo en la casilla [2, 2]. (A estas representaciones se les puede considerar *ad hoc*; los sistemas de bases de datos fueron desarrollados para proporcionar una manera más genérica, e independiente del dominio, de almacenar y recuperar hechos.) De lo que carecen los lenguajes de programación, es de algún mecanismo general para derivar hechos de otros hechos; cada actualización de la estructura de datos se realiza mediante un procedimiento específico del dominio, cuyos detalles se derivan del conocimiento acerca del dominio del o de la programadora. Este enfoque **procedural** puede contrastar con la naturaleza declarativa de la lógica proposicional, en la que el conocimiento y la inferencia se encuentran separados, y en la que la inferencia se realiza de forma totalmente independiente del dominio.

Otro inconveniente de las estructuras de datos de los programas (y de las bases de datos, respecto a este tema) es la falta de un mecanismo sencillo para expresar, por ejemplo, «Hay un hoyo en la casilla [2, 2] o en la [3, 1]» o «Si hay un *wumpus* en la casilla [1, 1] entonces no hay ninguno en la [2, 2]». Los programas pueden almacenar un valor único para cada variable, y algunos sistemas permiten el valor «desconocido», pero carecen de la expresividad que se necesita para manejar información incompleta.

La lógica proposicional es un lenguaje declarativo porque su semántica se basa en la relación de verdad entre las sentencias y los mundos posibles. Lo que tiene el suficiente poder expresivo para tratar información incompleta, mediante la disyunción y la conjunción. La lógica proposicional presenta una tercera característica que es muy deseable en los lenguajes de representación, a saber, la **composicionalidad**. En un lenguaje composicional, el significado de una sentencia es una función del significado de sus partes. Por ejemplo, «$M_{1,4} \wedge M_{1,2}$» está relacionada con los significados de «$M_{1,4}$» y «$M_{1,2}$». Sería muy extraño que «$M_{1,4}$» significara que hay mal hedor en la casilla [1, 4], que «$M_{1,2}$» significara que hay mal hedor en la casilla [1, 2], y que en cambio, «$M_{1,4} \wedge M_{1,2}$» significara que Francia y Polonia empataran en el partido de jockey de calificación de la última semana. Está claro que la no composicionalidad repercute en que al sistema de razonamiento le sea mucho más difícil subsistir.

COMPOSICIONAL

Tal como vimos en el Capítulo 7, la lógica proposicional carece del poder expresivo para describir de forma *precisa* un entorno con muchos objetos. Por ejemplo, estábamos forzados a escribir reglas separadas para cada casilla al hablar acerca de las brisas y de los hoyos, tal como

$$B_{1,1} \quad \Leftrightarrow \quad (H_{1,2} \vee H_{2,1})$$

Por otro lado, en el lenguaje natural parece bastante sencillo decir, de una vez por todas, que «En las casillas adyacentes a hoyos se percibe una pequeña brisa». De alguna manera, la sintaxis y la semántica del lenguaje natural nos hace posible describir el entorno de forma precisa.

De hecho, si lo pensamos por un momento, los lenguajes naturales (como el inglés o el castellano) son muy expresivos. Hemos conseguido escribir casi la totalidad de este libro en lenguaje natural, sólo con lapsos ocasionales en otros lenguajes (incluyendo la lógica, las matemáticas, y los lenguaje de diagramas visuales). En la lingüística y la filosofía del lenguaje hay una larga tradición que ve el lenguaje natural esencialmente como un lenguaje declarativo de representación del conocimiento e intenta definir su semántica formal. Como en un programa de investigación, si tuviera éxito sería de gran valor para la inteligencia artificial, porque esto permitiría utilizar un lenguaje natural (o alguna variación) con los sistemas de representación y razonamiento.

El punto de vista actual sobre el lenguaje natural es que sirve para un propósito algo diferente, a saber, como un medio de **comunicación** más que como una pura representación. Cuando una persona señala y dice, «¡Mira!», el que le oye llega a saber que lo que dice es que Superman finalmente ha aparecido sobre los tejados. Con ello, no queremos decir que la sentencia «¡Mira!» expresa ese hecho. Más bien, que el significado de la sentencia depende tanto de la propia sentencia como del **contexto** al que la sentencia hace referencia. Está claro que uno no podría almacenar una sentencia como «¡Mira!» en una base de conocimiento y esperar recuperar su significado sin haber almacenado también una representación de su contexto, y esto revela la problemática de cómo se puede representar el propio contexto. Los lenguajes naturales también son composicionales (el significado de una sentencia como «Entonces ella lo vio» puede depender de un contexto construido a partir de muchas sentencias precedentes y posteriores a ella). Por último, los lenguajes naturales sufren de la **ambigüedad**, que puede causar ciertos obstáculos en su comprensión. Tal como comenta Pinker (1995): «Cuando la gente piensa acerca de la *primavera*, seguramente no se confunden sobre si piensan acerca de una estación o algo que va ¿boing? (y si una palabra se puede corresponder con dos pensamientos, los pensamientos no pueden ser palabras).»

Nuestro enfoque consistirá en adoptar los fundamentos de la lógica proposicional (una semántica composicional declarativa que es independiente del contexto y no ambigua) y construir una lógica más expresiva basada en dichos fundamentos, tomando prestadas de los lenguajes naturales las ideas acerca de la representación, al mismo tiempo que evitando sus inconvenientes. Cuando observamos la sintaxis del lenguaje natural, los elementos más obvios son los nombres y las sentencias nominales que se refieren a los **objetos** (casillas, hoyos, *wumpus*) y los verbos y las sentencias verbales que se refieren a las **relaciones** entre los objetos (en la casilla se percibe una brisa, la casilla es adyacente a, el agente lanza una flecha). Algunas de estas relaciones son **funciones** (relaciones en las que dada una «entrada» se obtiene un solo «valor»). Es fácil empezar a listar ejemplos de objetos, relaciones y funciones:

OBJETOS

RELACIONES

FUNCIONES

- Objetos: gente, casas, números, teorías, Ronald McDonald, colores, partidos de béisbol, guerras, siglos…

PROPIEDADES

- Relaciones: éstas pueden ser relaciones unitarias, o **propiedades**, como ser de color rojo, ser redondo, ser ficticio, ser un número primo, ser multihistoriado…, o relaciones *n*-arias más generales, como ser hermano de, ser más grande que, estar dentro de, formar parte de, tener color, ocurrir después de, ser dueño de uno mismo, o interponerse entre…
- Funciones: el padre de, el mejor amigo de, el tercer turno, uno mayor que, el comienzo de…

Efectivamente, se puede pensar en casi cualquier aserción como una referencia a objetos y propiedades o relaciones. Como los siguientes ejemplos:

- «Uno sumado a dos es igual a tres.»
  Objetos: uno, dos, tres, uno sumado a dos; Relaciones: es igual a; Funciones: sumado a. («Uno sumado a dos» es el nombre de un objeto que se obtiene aplicando la función «sumado a» a los objetos «uno» y «dos». Hay otro nombre para este objeto.)
- «Las casillas que rodean al *wumpus* son pestilentes.»
  Objetos: *wumpus*, casillas; Propiedad: pestilente; Relación: rodear a.
- «El malvado rey Juan gobernó Inglaterra en 1200.»
  Objetos: Juan, Inglaterra, 1200; Relación: gobernar; Propiedades: malvado, rey.

El lenguaje de la **lógica de primer orden**, cuya sintaxis y semántica definiremos en la siguiente sección, está construido sobre objetos y relaciones. Precisamente por este motivo ha sido tan importante para las Matemáticas, la Filosofía y la inteligencia artificial (y en efecto, en el día a día de la existencia humana) porque se puede pensar en ello de forma utilitaria como en el tratamiento con objetos y de las relaciones entre éstos. La lógica de primer orden también puede expresar hechos acerca de *algunos* o *todos* los objetos de un universo de discurso. Esto nos permite representar leyes generales o reglas, tales como el enunciado «Las casillas que rodean al *wumpus* son pestilentes».

COMPROMISO ONTOLÓGICO

La principal diferencia entre la lógica proposicional y la de primer orden se apoya en el **compromiso ontológico** realizado por cada lenguaje (es decir, lo que asume cada uno acerca de la naturaleza de la *realidad*). Por ejemplo, la lógica proposicional asume que hay hechos que suceden o no suceden en el mundo. Cada hecho puede estar en uno de los dos estados: verdadero o falso[2]. La lógica de primer orden asume mucho más, a saber, que el mundo se compone de objetos con ciertas relaciones entre ellos que suceden o no suceden. Las lógicas de propósito específico aún hacen compromisos ontológicos más allá; por ejemplo, la **lógica temporal** asume que los hechos suceden en *tiempos* concretos y que esos tiempos (que pueden ser instantes o intervalos) están ordenados. De esta manera, las lógicas de propósito específico dan a ciertos tipos de objetos (y a los axiomas acerca de ellos) un estatus de «primera clase» dentro de la lógica, más que simplemente definiéndolos en la base de conocimiento. La **lógica de orden superior** ve las relaciones y funciones que se utilizan en la lógi-

LÓGICA TEMPORAL

LÓGICA DE ORDEN SUPERIOR

[2] A diferencia de los hechos en la **lógica difusa**, que tienen un **grado de verdad** entre 0 y 1. Por ejemplo, la sentencia «Vietnam es una gran ciudad» podría ser verdadera sólo con un grado 0,6 en nuestro mundo.

EL LENGUAJE DEL PENSAMIENTO

Los filósofos y los psicólogos han reflexionado profundamente sobre cómo representan el conocimiento los seres humanos y otros animales. Está claro que la evolución del lenguaje natural ha jugado un papel importante en el desarrollo de esta habilidad en los seres humanos. Por otro lado, muchas evidencias en la Psicología sugieren que los seres humanos no utilizan el lenguaje de forma directa en sus representaciones internas. Por ejemplo ¿cuál de las dos siguientes frases formaba el inicio de la Sección 8.1?

«En esta sección, discutiremos acerca de la naturaleza de los lenguajes de representación…»

«Esta sección cubre el tema de los lenguajes de representación del conocimiento…»

Wanner (1974) encontró que los sujetos hacían la elección acertada en los tests a un nivel casual (cerca del 50 por ciento de las veces) pero que recordaban el contenido de lo que habían leído con un 90 por ciento de precisión. Esto sugiere que la gente procesa las palabras para formar algún tipo de representación no verbal, lo que llamamos **memoria**.

El mecanismo concreto mediante el cual el lenguaje permite la representación y modela las ideas en los seres humanos sigue siendo una incógnita fascinante. La famosa hipótesis de **Sapir-Whorf** sostiene que el lenguaje que hablamos influye profundamente en la manera en que pensamos y tomamos decisiones, en concreto, estableciendo las estructuras de categorías con las que separamos el mundo en diferentes agrupaciones de objetos. Whorf (1956) sostuvo que los esquimales tenían muchas palabras para la nieve, así que tenían una experiencia de la nieve diferente de las personas que hablaban otros idiomas. Algunos lingüistas no están de acuerdo con el fundamento factual de esta afirmación (Pullum (1991) argumenta que el Inuit, el Yupik, y otros lenguajes similares parece que tienen un número parecido de palabras que el inglés para los conceptos relacionados con la nieve) mientras que otros apoyan dicha afirmación (Fortescue, 1984). Parece perfectamente comprensible que las poblaciones que tienen una familiaridad mayor con algunos aspectos del mundo desarrollan un vocabulario mucho más detallado en dichos temas, por ejemplo, los entomólogos dividen lo que muchos de nosotros llamamos simplemente *escarabajos* en cientos de miles de especies y además están personalmente familiarizados con muchas de ellas. (El biólogo evolucionista J. B. S. Haldane una vez se quejó de «Una afición desmesurada a los escarabajos» por parte del Creador.) Más aún, los esquiadores expertos tienen muchos términos para la nieve (en polvo, sopa de pescado, puré de patatas, cruda, maíz, cemento, pasta, azúcar, asfalto, pana, pelusa, etcétera) que representan diferencias que a los profanos no nos son familiares. Lo que no está claro es la dirección de la causalidad (¿los esquiadores se dan cuenta de las diferencias sólo por aprender las palabras, o las diferencias surgen de la experiencia individual y llegan a emparejarse con las etiquetas que se están utilizando en el grupo?) Esta cuestión es especialmente importante en el estudio del desarrollo infantil. Hasta ahora disponemos de poco entendimiento acerca del grado en el cual el aprendizaje del lenguaje y el razonamiento están entrelazados. Por ejemplo, ¿el conocimiento del nombre de un concepto, como *licenciado*, hace que nos sea más fácil construir y razonar acerca de conceptos más complejos que engloban a dicho nombre, como *licenciado idóneo*?

ca de primer orden como objetos en sí mismos. Esto nos permite hacer aserciones acerca de *todas* las relaciones, por ejemplo, uno podría desear definir lo que significa que una relación sea transitiva. A diferencia de las lógicas de propósito específico, la lógica de orden superior es estrictamente más expresiva que la lógica de primer orden, en el sentido de que algunas sentencias de la lógica de orden superior no se pueden expresar mediante un número finito de sentencias de la lógica de primer orden.

COMPROMISOS EPISTEMOLÓGICOS

Una lógica también se puede caracterizar por sus **compromisos epistemológicos** (los posibles estados del conocimiento respecto a cada hecho que la lógica permite). Tanto en la lógica proposicional como en la de primer orden, una sentencia representa un hecho y el agente o bien cree que la sentencia es verdadera, o cree que la sentencia es falsa, o no tiene ninguna opinión. Por lo tanto, estas lógicas tienen tres estados posibles de conocimiento al considerar cualquier sentencia. Por otro lado, los sistemas que utilizan la **teoría de las probabilidades** pueden tener un *grado de creencia*, que va de cero (no se cree en absoluto) a uno (se tiene creencia total)[3]. Por ejemplo, un agente del mundo de *wumpus* que utilice las probabilidades podría creer que el *wumpus* se encuentra en la casilla [1, 3] con una probabilidad de 0,75. En la Figura 8.1 se resumen los compromisos ontológicos y epistemológicos de cinco lógicas distintas.

En la siguiente sección nos meteremos en los detalles de la lógica de primer orden. Al igual que un estudiante de Física necesita familiarizarse con las Matemáticas, un estudiante de IA debe desarrollar sus capacidades para trabajar con la notación lógica. Por otro lado, *no* es tan importante conseguir una preocupación desmesurada sobre las *especificaciones* de la notación lógica (al fin y al cabo, hay docenas de versiones distintas). Los conceptos principales que hay que tener en cuenta son cómo el lenguaje nos facilita una representación precisa y cómo su semántica nos permite realizar procedimientos sólidos de razonamiento.

Lenguaje	Compromiso ontológico (lo que sucede en el mundo)	Compromiso epistemológico (lo que el agente cree acerca de los hechos)
Lógica proposicional	Hechos	Verdadero/falso/desconocido
Lógica de primer orden	Hechos, objetos, relaciones	Verdadero/falso/desconocido
Lógica temporal	Hechos, objetos, relaciones, tiempos	Verdadero/falso/desconocido
Teoría de las probabilidades	Hechos	Grado de creencia $\in$ [0, 1]
Lógica difusa	Hechos con un grado de verdad $\in$ [0, 1]	Valor del intervalo conocido

Figura 8.1 Lenguajes formales y sus compromisos ontológicos y epistemológicos.

[3] Es importante no confundir el grado de creencia de la teoría de probabilidades con el grado de verdad de la lógica difusa. Realmente, algunos sistemas difusos permiten una incertidumbre (un grado de creencia) acerca de los grados de verdad.

8.2 Sintaxis y semántica de la lógica de primer orden

Comenzamos esta sección especificando de forma más precisa la forma en la que los mundos posibles en la lógica de primer orden reflejan el compromiso ontológico respecto a los objetos y las relaciones. Entonces introducimos los diferentes elementos del lenguaje, explicando su semántica a medida que avanzamos.

Modelos en lógica de primer orden

Recuerde del Capítulo 7 que los modelos en un lenguaje lógico son las estructuras formales que establecen los mundos posibles que se tienen en cuenta. Los modelos de la lógica proposicional son sólo conjuntos de valores de verdad para los símbolos proposicionales. Los modelos de la lógica de primer orden son más interesantes. Primero, ¡éstos contienen a los objetos! El **dominio** de un modelo es el conjunto de objetos que contiene; a estos objetos a veces se les denomina **elementos del dominio**. La Figura 8.2 muestra un modelo con cinco objetos: Ricardo Corazón de León, Rey de Inglaterra de 1189 a 1199; su hermano más joven, el malvado Rey Juan, quien reinó de 1199 a 1215; las piernas izquierda de Ricardo y Juan; y una corona.

DOMINIO

ELEMENTOS DEL DOMINIO

Los objetos en el modelo pueden estar relacionados de diversas formas. En la figura, Ricardo y Juan son hermanos. Hablando formalmente, una relación es sólo un con-

Figura 8.2 Un modelo que contiene cinco objetos, dos relaciones binarias, tres relaciones unitarias (indicadas mediante etiquetas sobre los objetos), y una función unitaria: pierna izquierda.

TUPLAS

junto de **tuplas** de objetos que están relacionados. (Una tupla es una colección de objetos colocados en un orden fijo que se escriben entre paréntesis angulares.) De esta manera, la relación de hermandad en este modelo es el conjunto

$$\{\langle \text{Ricardo Corazón de León, Rey Juan}\rangle, \langle \text{Rey Juan, Ricardo Corazón de León}\rangle\} \quad (8.1)$$

(Aquí hemos nombrado los objetos en español, pero se puede, si uno lo desea, sustituir mentalmente los nombres por las imágenes.) La corona está colocada sobre la cabeza del Rey Juan, así que la relación «sobre la cabeza» contiene sólo una tupla, ⟨Corona, Rey Juan⟩. Las relaciones «hermano» y «sobre la cabeza» son relaciones binarias, es decir, relacionan parejas de objetos. El modelo también contiene relaciones unitarias, o propiedades: la propiedad «ser persona» es verdadera tanto para Ricardo como para Juan; la propiedad «ser rey» es verdadera sólo para Juan (presumiblemente porque Ricardo está muerto en este instante); y la propiedad «ser una corona» sólo es verdadera para la corona.

Hay ciertos tipos de relaciones que es mejor que se consideren como funciones; en estas relaciones un objeto dado debe relacionarse exactamente con otro objeto. Por ejemplo, cada persona tiene una pierna izquierda, entonces el modelo tiene la función unitaria «pierna izquierda» con las siguientes aplicaciones:

$$\begin{aligned}&\langle \text{Ricardo Corazón de León}\rangle \rightarrow \text{pierna izquierda de Ricardo}\\&\langle \text{Rey Juan}\rangle \rightarrow \text{pierna izquierda de Juan}\end{aligned} \quad (8.2)$$

FUNCIONES TOTALES

Hablando de forma estricta, los modelos en la lógica de primer orden requieren **funciones totales**, es decir, debe haber un valor para cada tupla de entrada. Así, la corona debe tener una pierna izquierda y por lo tanto, también cada una de las piernas izquierdas. Hay una solución técnica para este problema inoportuno, incluyendo un objeto «invisible» adicional, que es la pierna izquierda de cada cosa que no tiene pierna izquierda, inclusive ella misma. Afortunadamente, con tal de que uno no haga aserciones acerca de piernas izquierdas de cosas que no tienen piernas izquierdas, estos tecnicismos dejan de tener importancia.

Símbolos e interpretaciones

Ahora volvemos a la sintaxis del lenguaje. El lector impaciente puede obtener una descripción completa de la gramática formal de la lógica de primer orden en la Figura 8.3.

Los elementos sintácticos básicos de la lógica de primer orden son los símbolos que representan los objetos, las relaciones y las funciones. Por consiguiente, los símbolos se agrupan en tres tipos: **símbolos de constante**, que representan objetos; **símbolos de predicado**, que representan relaciones; y **símbolos de función**, que representan funciones. Adoptamos la convención de que estos símbolos comiencen en letra mayúscula. Por ejemplo, podríamos utilizar los símbolos de constante *Ricardo* y *Juan*; los símbolos de predicado *Hermano*, *SobreCabeza*, *Persona*, *Rey* y *Corona*; y el símbolo de función *PiernaIzquierda*. Al igual que con los símbolos proposicionales, la selección de los nombres depende enteramente del usuario. Cada símbolo de predicado y de función tiene una **aridad** que establece su número de argumentos.

SÍMBOLOS DE CONSTANTE

SÍMBOLOS DE PREDICADO

SÍMBOLOS DE FUNCIÓN

ARIDAD

$$
\begin{array}{rcl}
\textit{Sentencia} & \rightarrow & \textit{SentenciaAtómica} \\
 & | & (\textit{Sentencia Conectiva Sentencia}) \\
 & | & \textit{Cuantificador Variable}\ldots\ \textit{Sentencia} \\
 & | & \neg\textit{Sentencia} \\
 & & \\
\textit{SentenciaAtómica} & \rightarrow & \textit{Predicado}(\textit{Término}\ldots) \mid \textit{Término} = \textit{Término} \\
 & & \\
\textit{Término} & \rightarrow & \textit{Función}(\textit{Término}\ldots) \\
 & | & \textit{Constante} \\
 & | & \textit{Variable} \\
 & & \\
\textit{Conectiva} & \rightarrow & \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
\textit{Cuantificador} & \rightarrow & \forall \mid \exists \\
\textit{Constante} & \rightarrow & A \mid X_1 \mid \textit{Juan} \mid \cdots \\
\textit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\textit{Predicado} & \rightarrow & \textit{AntesDe} \mid \textit{TieneColor} \mid \textit{EstáLLoviendo} \mid \cdots \\
\textit{Función} & \rightarrow & \textit{Madre} \mid \textit{PiernaIzquierda} \mid \cdots
\end{array}
$$

Figura 8.3 La sintaxis de la lógica de primer orden con igualdad, espicificada en BNF. (Mire la página 984 si no estás familiarizado con esta notación.) La sintaxis es estricta con el tema de los paréntesis; los comentarios acerca de los paréntesis y la precedencia de los operadores de la página 230 se aplica de la misma forma a la lógica de primer orden.

INTERPRETACIÓN

INTERPRETACIÓN DESEADA

La semántica debe relacionar las sentencias con los modelos para determinar su valor de verdad. Para que esto ocurra, necesitamos de una **interpretación** que especifique exactamente qué objetos, relaciones y funciones son referenciados mediante símbolos de constante, de predicados y de función, respectivamente. Una interpretación posible para nuestro ejemplo (a la que llamaremos **interpretación deseada**) podría ser la siguiente:

- *Ricardo* se refiere a Ricardo Corazón de León y *Juan* se refiere al malvado Rey Juan.
- *Hermano* se refiere a la relación de hermandad, es decir, al conjunto de tuplas de objetos que se muestran en la Ecuación (8.1); *SobreCabeza* se refiere a la relación «sobre la cabeza» que sucede entre la corona y el Rey Juan; *Persona*, *Rey* y *Corona* se refieren a los conjuntos de objetos que son personas, reyes y coronas.
- *PiernaIzquierda* se refiere a la función «pierna izquierda», es decir, la aplicación que se muestra en la Ecuación (8.2).

Hay muchas otras interpretaciones posibles que se relacionan con estos símbolos para este modelo en concreto. Por ejemplo, una interpretación podría relacionar *Ricardo* con la corona y *Juan* con la pierna izquierda del Rey Juan. Hay cinco objetos, por lo tanto hay 25 interpretaciones posibles sólo para los símbolos de constante *Ricardo* y *Juan*. Fíjese en que no todos los objetos necesitan un nombre (por ejemplo, la interpretación deseada no nombra la corona o las piernas). También es posible que un objeto tenga varios

nombres; hay una interpretación en la que tanto *Ricardo* como *Juan* se refieren a la corona. Si encuentra que le confunde esta posibilidad recuerde que en la lógica proposicional es totalmente posible tener un modelo en el que *Nublado* y *Soleado* sean ambos verdaderos; la tarea de la base de conocimiento consiste justamente en excluir lo que es inconsistente con nuestro conocimiento.

El valor de verdad de cualquier sentencia se determina por un modelo y por una interpretación de los símbolos de la sentencia. Por lo tanto, la implicación, la validez, etcétera, se determinan con base en *todos los modelos posibles* y *todas las interpretaciones posibles*. Es importante fijarse en que el número de elementos del dominio en cada modelo puede ser infinito, por ejemplo, los elementos del dominio pueden ser números enteros o reales. Por eso, el número de modelos posibles es infinito, igual que el número de interpretaciones. La comprobación de la implicación mediante la enumeración de todos los modelos posibles, que funcionaba en la lógica proposicional, no es una opción acertada para la lógica de primer orden. Aunque el número de objetos esté restringido, el número de combinaciones puede ser enorme. Con los símbolos de nuestro ejemplo, hay aproximadamente 10^{25} combinaciones para un dominio de cinco objetos. (*Véase* Ejercicio 8.5.)

Términos

TÉRMINO

Un **término** es una expresión lógica que se refiere a un objeto. Por lo tanto, los símbolos de constante son términos, pero no siempre es conveniente tener un símbolo distinto para cada objeto. Por ejemplo, en español podríamos utilizar la expresión «la pierna izquierda del Rey Juan», y sería mucho mejor que darle un nombre a su pierna. Para esto sirven los símbolos de función: en vez de utilizar un símbolo de constante utilizamos *PiernaIzquierda*(*Juan*). En el caso general, un término complejo está formado por un símbolo de función seguido de una lista de términos entre paréntesis que son los argumentos del símbolo de función. Es importante recordar que un término complejo tan sólo es un tipo de nombre algo complicado. No es una «llamada a una subrutina» que «devuelve un valor». No hay una subrutina *PiernaIzquierda* que tome una persona como entrada y devuelva una pierna. Podemos razonar acerca de piernas izquierdas (por ejemplo haciendo constar que cada uno tiene una pierna y entonces deducir que Juan debe tener una) sin tener que proporcionar una definición de *PiernaIzquierda*. Esto es algo que no se puede hacer mediante subrutinas en los lenguajes de programación[4].

La semántica formal de los términos es sencilla. Considera un término $f(t_1 \ldots, t_n)$. El símbolo de función f se refiere a alguna función del modelo (llamémosla F); los términos argumento se refieren a objetos del dominio (llamémoslos $d_1 \ldots, d_n$); y el término en su globalidad se refiere al objeto que es el valor obtenido de aplicar la función F a los

[4] Las **expresiones-λ** proporcionan una notación útil mediante la cual se construyen nuevos símbolos de función «al vuelo». Por ejemplo, la función que eleva al cuadrado su argumento se puede escribir como ($\lambda x\ x \times x$) y se puede aplicar a argumentos del mismo modo que cualquier otro símbolo de función. Una expresión-λ también se puede definir y utilizar como un símbolo de predicado. (*Véase* Capítulo 22.) El operador `lambda` del Lisp juega exactamente el mismo papel. Fíjese en que el uso de λ de este modo *no* aumenta el poder expresivo de la lógica de primer orden; porque cualquier sentencia que tenga una expresión-λ se puede rescribir «enchufando» sus argumentos para obtener una sentencia equivalente.

objetos $d_1 \ldots, d_n$. Por ejemplo, supongamos que el símbolo de función *PiernaIzquierda* se refiere a la función que se muestra en la Ecuación (8.2) y que Juan se refiere al Rey Juan, entonces, *PiernaIzquierda*(*Juan*) se refiere a la pierna izquierda del Rey Juan. De esta manera, la interpretación establece el referente de cada término.

Sentencias atómicas

Ahora que ya tenemos tanto los términos para referirnos a los objetos, como los símbolos de predicado para referirnos a las relaciones, podemos juntarlos para construir **sentencias atómicas** que representan hechos. Una sentencia atómica está compuesta por un símbolo de predicado seguido de una lista de términos entre paréntesis:

$$Hermano(Ricardo, Juan)$$

Esto representa, bajo la interpretación deseada que hemos dado antes, que Ricardo Corazón de León es el hermano del Rey Juan[5]. Las sentencias atómicas pueden tener términos complejos. De este modo,

$$CasadoCon(Padre(Ricardo), Madre(Juan))$$

representa que el padre de Ricardo Corazón de León está casado con la madre del Rey Juan (otra vez, bajo la adecuada interpretación).

Una sentencia atómica es ***verdadera*** *en un modelo dado, y bajo una interpretación dada, si la relación referenciada por el símbolo de predicado sucede entre los objetos referenciados por los argumentos.*

Sentencias compuestas

Podemos utilizar las **conectivas lógicas** para construir sentencias más complejas, igual que en la lógica proposicional. La semántica de las sentencias formadas con las conectivas lógicas es idéntica a la de la lógica proposicional. Aquí hay cuatro sentencias que son verdaderas en el modelo de la Figura 8.2, bajo la interpretación deseada:

$$\neg Hermano(PiernaIzquierda(Ricardo), Juan)$$
$$Hermano(Ricardo, Juan) \wedge Hermano(Juan, Ricardo)$$
$$Rey(Ricardo) \vee Rey(Juan)$$
$$\neg Rey(Ricardo) \Rightarrow Rey(Juan)$$

Cuantificadores

CUANTIFICADORES

Una vez tenemos una lógica que nos permite representar objetos, es muy natural querer expresar las propiedades de colecciones enteras de objetos en vez de enumerar los objetos por su nombre. Los **cuantificadores** nos permiten hacer esto. La lógica de primer orden contiene dos cuantificadores estándar, denominados *universal* y *existencial*.

[5] Por lo general utilizaremos la convención de ordenación de los argumentos $P(x, y)$, que se interpreta como «x es P de y».

Cuantificador universal (∀)

Retomemos la dificultad que teníamos en el Capítulo 7 con la expresión de las reglas generales en la lógica proposicional. Las reglas como «Las casillas vecinas al *wumpus* son apestosas» y «Todos los reyes son personas» son el pan de cada día de la lógica de primer orden. En la Sección 8.3 trataremos con la primera de éstas. Respecto a la segunda regla, «Todos los reyes son personas», se escribe en la lógica de primer orden

$$\forall x \;\; Rey(x) \;\Rightarrow\; Persona(x)$$

generalmente ∀ se pronuncia «Para todo…». (Recuerda que la A boca abajo representa «todo».) Así, la sentencia dice, «Para todo x, si x es un rey, entonces x es una persona». Al símbolo x se le llama **variable**. Por convenio, las variables se escriben en minúsculas. Una variable es un término en sí mismo, y como tal, también puede utilizarse como el argumento de una función, por ejemplo, $PiernaIzquierda(x)$. Un término que no tiene variables se denomina **término base**.

VARIABLE

TÉRMINO BASE

De forma intuitiva, la sentencia $\forall x\, P$, donde P es una expresión lógica, dice que P es verdadera para cada objeto x. Siendo más precisos, $\forall x\, P$ es verdadera en un modelo dado bajo una interpretación dada, si P es verdadera para todas las **interpretaciones ampliadas**, donde cada interpretación ampliada especifica un elemento del dominio al que se refiere x.

INTERPRETACIÓN AMPLIADA

Esto suena algo complicado, pero tan sólo es una manera cautelosa de definir el sentido intuitivo de la cuantificación universal. Considere el modelo que se muestra en la Figura 8.2 y la interpretación deseada que va con él. Podemos ampliar la interpretación de cinco maneras:

$x \rightarrow$ Ricardo Corazón de León,
$x \rightarrow$ Rey Juan,
$x \rightarrow$ pierna izquierda de Ricardo,
$x \rightarrow$ pierna izquierda de Juan,
$x \rightarrow$ la corona.

La sentencia cuantificada universalmente $\forall x \;\; Rey(x) \Rightarrow Persona(x)$ es verdadera bajo la interpretación inicial si la sentencia $Rey(x) \Rightarrow Persona(x)$ es verdadera en cada una de las interpretaciones ampliadas. Es decir, la sentencia cuantificada universalmente es equivalente a afirmar las cinco sentencias siguientes:

Ricardo Corazón de León es un rey ⇒ Ricardo Corazón de León es una persona.
Rey Juan es un rey ⇒ Rey Juan es una persona.
La pierna izquierda de Ricardo es un rey ⇒ La pierna izquierda de Ricardo es una persona.
La pierna izquierda de Juan es un rey ⇒ La pierna izquierda de Juan es una persona.
La corona es un rey ⇒ La corona es una persona.

Vamos a observar cuidadosamente este conjunto de aserciones. Ya que en nuestro modelo el Rey Juan es el único rey, la segunda sentencia aserta que él es una persona, tal como esperamos. Pero, ¿qué ocurre con las otras cuatro sentencias, donde parece que incluso se reivindica acerca de piernas y coronas? ¿Eso forma parte del sentido que tie-

ne «Todos los reyes son personas»? De hecho, las otras cuatro aserciones son verdaderas en el modelo, pero no hacen en absoluto ninguna reivindicación acerca de la naturaleza de persona de las piernas, coronas, o en efecto de Ricardo. Esto es porque ninguno de estos objetos es un rey. Mirando la tabla de verdad de la conectiva $\Rightarrow$ (Figura 7.8) vemos que la implicación es verdadera siempre que su premisa sea falsa (*independientemente* del valor de verdad de la conclusión). Así que, al afirmar una sentencia cuantificada universalmente, que es equivalente a afirmar la lista total de implicaciones individuales, acabamos afirmando la conclusión de la regla sólo para aquellos objetos para los que la premisa es verdadera y no decimos nada acerca de aquellos individuos para los que la premisa es falsa. De este modo, las entradas para la tabla de verdad de la conectiva $\Rightarrow$ son perfectas para escribir reglas generales mediante cuantificadores universales.

Un error común, hecho frecuentemente aún por los lectores más diligentes que han leído este párrafo varias veces, es utilizar la conjunción en vez de la implicación. La sentencia

$$\forall x \;\; Rey(x) \wedge Persona(x)$$

sería equivalente a afirmar

Ricardo Corazón de León es un rey $\wedge$ Ricardo Corazón de León es una persona
Rey Juan es un rey $\wedge$ Rey Juan es una persona
La pierna izquierda de Ricardo es un rey $\wedge$ La pierna izquierda de Ricardo es una persona

etcétera. Obviamente, esto no plasma lo que queremos expresar.

Cuantificación existencial ($\exists$)

La cuantificación universal construye enunciados acerca de todos los objetos. De forma similar, utilizando un cuantificador existencial, podemos construir enunciados acerca de *algún* objeto del universo de discurso sin nombrarlo. Para decir, por ejemplo, que el Rey Juan tiene una corona sobre su cabeza, escribimos

$$\exists x \; Corona(x) \wedge SobreCabeza(x, Juan)$$

$\exists x$ se pronuncia «Existe un x tal que…» o «Para algún x…».

De forma intuitiva, la sentencia $\exists x \; P$ dice que P es verdadera al menos para un objeto x. Siendo más precisos, $\exists x \; P$ es verdadera en un modelo dado bajo una interpretación dada si P es verdadera *al menos en una* interpretación ampliada que asigna a x un elemento del dominio. Para nuestro ejemplo, esto significa que al menos una de las sentencias siguientes debe ser verdadera:

Ricardo Corazón de León es una corona $\wedge$ Ricardo Corazón de León está sobre la cabeza de Juan;
Rey Juan es una corona $\wedge$ Rey Juan está sobre la cabeza de Juan;
La pierna izquierda de Ricardo es una corona $\wedge$ La pierna izquierda de Ricardo está sobre la cabeza de Juan;

La pierna izquierda de Juan es una corona $\wedge$ La pierna izquierda de Juan está sobre la cabeza de Juan;
La corona es una corona $\wedge$ La corona está sobre la cabeza de Juan.

La quinta aserción es verdadera en nuestro modelo, por lo que la sentencia original cuantificada existencialmente es verdadera en el modelo. Fíjese en que, según nuestra definición, la sentencia también sería verdadera en un modelo en el que el Rey Juan llevara dos coronas. Esto es totalmente consistente con la sentencia inicial «El Rey Juan tiene una corona sobre su cabeza»[6].

Igual que el utilizar con el cuantificador $\forall$ la conectiva $\Rightarrow$ parece ser lo natural, $\wedge$ es la conectiva natural para ser utilizada con el cuantificador $\exists$. Utilizar $\wedge$ como la conectiva principal con $\forall$ nos llevó a un enunciado demasiado fuerte en el ejemplo de la sección anterior; y en efecto, utilizar $\Rightarrow$ con $\exists$ nos lleva a un enunciado demasiado débil. Considere la siguiente sentencia:

$$\exists x\ Corona(x) \Rightarrow SobreCabeza(x, Juan)$$

Superficialmente, esto podría parecer una interpretación razonable de nuestra sentencia. Al aplicar la semántica vemos que la sentencia dice que al menos una de las aserciones siguientes es verdadera:

Ricardo Corazón de León es una corona $\Rightarrow$ Ricardo Corazón de León está sobre la cabeza de Juan;
Rey Juan es una corona $\Rightarrow$ Rey Juan está sobre la cabeza de Juan;
La pierna izquierda de Ricardo es una corona $\Rightarrow$ La pierna izquierda de Ricardo está sobre la cabeza de Juan;

etcétera. Ahora una implicación es verdadera si son verdaderas la premisa y la conclusión, *o si su premisa es falsa*. Entonces, si Ricardo Corazón de León no es una corona, entonces la primera aserción es verdadera y se satisface el existencial. Así que, una implicación cuantificada existencialmente es verdadera en cualquier modelo que contenga un objeto para el que la premisa de la implicación sea falsa; de aquí que este tipo de sentencias al fin y al cabo no digan mucho.

Cuantificadores anidados

A menudo queremos expresar sentencias más complejas utilizando múltiples cuantificadores. El caso más sencillo es donde los cuantificadores son del mismo tipo. Por ejemplo, «Los camaradas son hermanos» se puede escribir como

$$\forall x\ \ \forall y\ \ Hermano(x, y) \Rightarrow Camarada(x, y)$$

[6] Hay una variante del cuantificador existencial, escrito por lo general $\exists^1$ o $\exists!$, que significa «Existe exactamente uno.» El mismo significado se puede expresar utilizando sentencias de igualdad, tal como mostraremos en esta misma sección.

Los cuantificadores consecutivos del mismo tipo se pueden escribir como un solo cuantificador con sendas variables. Por ejemplo, para decir que la relación de hermandad es una relación simétrica podemos escribir

$$\forall x, y \;\; Camarada(x, y) \Rightarrow Camarada(y, x)$$

En otros casos tenemos combinaciones. «Todo el mundo ama a alguien» significa que para todas las personas, hay alguien que esa persona ama:

$$\forall x \;\; \exists y \;\; Ama(x, y)$$

Por otro lado, para decir «Hay alguien que es amado por todos», escribimos

$$\exists y \;\; \forall x \;\; Ama(x, y)$$

Por lo tanto, el orden de los cuantificadores es muy importante. Está más claro si introducimos paréntesis. $\forall x\ (\exists y\ Ama(x, y))$ dice que *todo el mundo* tiene una propiedad en particular, en concreto, la propiedad de amar a alguien. Por otro lado, $\exists y\ (\forall y\ Ama(x, y))$ dice que *alguien* en el mundo tiene una propiedad particular, en concreto, la propiedad de ser amado por todos.

Puede aparecer alguna confusión cuando dos cuantificadores se utilizan con el mismo identificador de variable. Considere la sentencia

$$\forall x\ [Corona(x) \vee (\exists x\ Hermano(Ricardo, x))]$$

Aquí la x de $Hermano(Ricardo, x)$ está cuantificada existencialmente. La regla es que la variable pertenece al cuantificador más anidado que la mencione; entonces no será el sujeto de cualquier otro cuantificador[7]. Otra forma de pensar en esto es: $\exists x\ Hermano(Ricardo, x)$ es una sentencia acerca de Ricardo (que él tiene un hermano), no acerca de x; así que poner $\forall x$ fuera no tiene ningún efecto. Se podría perfectamente haber escrito $\exists z\ Hermano(Ricardo, z)$. Y como esto puede ser una fuente de confusión, siempre utilizaremos variables diferentes.

Conexiones entre ∀ y ∃

Los dos cuantificadores realmente están íntimamente conectados el uno al otro, mediante la negación. Afirmar que a todo el mundo no le gustan las pastinacas es lo mismo que afirmar que no existe alguien a quien le gusten, y viceversa:

$$\forall x \;\; \neg Gusta(x, Pastinacas) \quad \text{es equivalente a} \quad \neg\exists x \;\; Gusta(x, Pastinacas).$$

[7] Es el potencial para la inferencia entre cuantificadores que utilizan el mismo identificador de variable lo que motiva el mecanismo barroco de las interpretaciones ampliadas en la semántica de las sentencias cuantificadas. El enfoque intuitivo más obvio de sustituir los objetos de cada ocurrencia de x falla en nuestro ejemplo porque la x de $Hermano(Ricardo, x)$ sería «capturada» por la sustitución. Las interpretaciones ampliadas manejan este tema de forma correcta porque la asignación para x del cuantificador más interiorizado estropea a los cuantificadores externos.

Podemos dar un paso más allá: «A todo el mundo le gusta el helado» significa que no hay nadie a quien no le guste el helado:

$$\forall x \;\; Gusta(x, Helado) \quad \text{es equivalente a} \quad \neg\exists x \;\; \neg Gusta(x, Helado).$$

Como $\forall$ realmente es una conjunción sobre el universo de objetos y $\exists$ es una disyunción, no sería sorprendente que obedezcan a las leyes de Morgan. Las leyes de Morgan para las sentencias cuantificadas y no cuantificadas son las siguientes:

$$\begin{array}{lcl@{\qquad}lcl}
\forall x \;\; \neg P & \equiv & \neg\exists x \;\; P & \neg P \wedge \neg Q & \equiv & \neg(P \vee Q) \\
\neg\forall x \;\; P & \equiv & \exists x \;\; \neg P & \neg(P \wedge Q) & \equiv & \neg P \vee \neg Q \\
\forall x \;\; P & \equiv & \neg\exists x \;\; \neg P & P \wedge Q & \equiv & \neg(\neg P \vee \neg Q) \\
\exists x \;\; P & \equiv & \neg\forall x \;\; \neg P & P \vee Q & \equiv & \neg(\neg P \wedge \neg Q)
\end{array}$$

De este modo, realmente no necesitamos $\forall$ y $\exists$ al mismo tiempo, igual que no necesitamos $\wedge$ y $\vee$ al mismo tiempo. Todavía es más importante la legibilidad que la parquedad, así que seguiremos utilizando ambos cuantificadores.

Igualdad

SÍMBOLO DE IGUALDAD

La lógica de primer orden incluye un mecanismo extra para construir sentencias atómicas, uno que no utiliza un predicado y unos términos como hemos descrito antes. En lugar de ello, podemos utilizar el **símbolo de igualdad** para construir enunciados describiendo que dos términos se refieren al mismo objeto. Por ejemplo,

$$Padre(Juan) = Enrique$$

dice que el objeto referenciado por *Padre*(*Juan*) y el objeto referenciado por *Enrique* son el mismo. Como una interpretación especifica el referente para cualquier término, determinar el valor de verdad de una sentencia de igualdad consiste simplemente en ver que los referentes de los dos términos son el mismo objeto.

El símbolo de igualdad se puede utilizar para representar hechos acerca de una función dada, tal como hicimos con el símbolo *Padre*, también se puede utilizar con la negación para insistir en que dos términos no son el mismo objeto. Para decir que Ricardo tiene al menos dos hermanos escribiríamos

$$\exists x, y \; Hermano(x, Ricardo) \wedge Hermano(y, Ricardo) \wedge \neg(x = y)$$

La sentencia

$$\exists x, y \; Hermano(x, Ricardo) \wedge Hermano(y, Ricardo)$$

no tiene el significado deseado. En concreto, es verdadero en el modelo de la Figura 8.2, donde Ricardo tiene sólo un hermano. Para verlo, considere las interpretaciones ampliadas en las que x e y son asignadas al Rey Juan. La adición de $\neg(x = y)$ excluye dichos modelos. La notación $x \neq y$ se utiliza a veces como abreviación de $\neg(x = y)$.

8.3 Utilizar la lógica de primer orden

DOMINIOS

Ahora que hemos definido un lenguaje lógico expresivo, es hora de aprender a utilizarlo. La mejor forma de hacerlo es a través de ejemplos. Hemos visto algunas sentencias sencillas para mostrar los diversos aspectos de la sintaxis lógica; en esta sección proporcionaremos unas representaciones más sistemáticas de algunos **dominios** sencillos. En la representación del conocimiento un dominio es sólo algún ámbito del mundo acerca del cual deseamos expresar algún conocimiento.

Comenzaremos con una breve descripción de la interfaz DECIR/PREGUNTAR para las bases de conocimiento en primer orden. Entonces veremos los dominios de las relaciones de parentesco, de los números, de los conjuntos, de las listas y del mundo de *wumpus*. La siguiente sección contiene un ejemplo mucho más sustancial (sobre circuitos electrónicos) y en el Capítulo 10 cubriremos cada aspecto del universo de discurso.

Aserciones y peticiones en lógica de primer orden

AFIRMACIONES

Las sentencias se van añadiendo a la base de conocimiento mediante DECIR, igual que en la lógica proposicional. Este tipo de sentencias se denominan **aserciones**. Por ejemplo, podemos afirmar que Juan es un rey y que los reyes son personas mediante las siguientes sentencias:

$$\text{DECIR}(BC,\ Rey(Juan))$$
$$\text{DECIR}(BC,\ \forall x\ \ Rey(x) \Rightarrow Persona(x))$$

Podemos hacer preguntas a la base de conocimiento mediante PREGUNTAR. Por ejemplo,

$$\text{PREGUNTAR}(BC, Rey(Juan))$$

PETICIONES

OBJETIVOS

que devuelve *verdadero*. Las preguntas realizadas con PREGUNTAR se denominan **peticiones** u **objetivos** (no deben confundirse con los objetivos que se utilizan para describir los estados deseados por el agente). En general, cualquier petición que se implica lógicamente de la base de conocimiento sería respondida afirmativamente. Por ejemplo, dadas las dos aserciones en el párrafo precedente, la petición

$$\text{PREGUNTAR}(BC, Persona(Juan))$$

también devolvería *verdadero*. También podemos realizar peticiones cuantificadas, tales como

$$\text{PREGUNTAR}(BC, \exists x\ Persona(x)).$$

SUSTITUCIÓN

LISTA DE LIGADURAS

La respuesta a esta petición podría ser *verdadero*, pero esto no es de ayuda ni es ameno. (Es como responder a «¿Me puedes decir qué hora es?» con un «Sí».) Una petición con variables existenciales es como preguntar «Hay algún x tal qué…» y lo resolvemos proporcionando dicha x. La forma estándar para una respuesta de este tipo es una **sustitución** o **lista de ligaduras**, que es un conjunto de parejas de variable/término. En este

caso en particular, dadas las dos aserciones, la respuesta sería $\{x/Juan\}$. Si hay más de una respuesta posible se puede devolver una lista de sustituciones.

El dominio del parentesco

El primer ejemplo que vamos a tratar es el dominio de las relaciones familiares, o de parentesco. Este dominio incluye hechos como «Isabel es la madre de Carlos» y «Carlos es el padre de Guillermo», y reglas como «La abuela de uno es la madre de su padre».

Está claro que los objetos de nuestro dominio son personas. Tendremos dos predicados unitarios: *Masculino* y *Femenino*. Las relaciones de parentesco (de paternidad, de hermandad, de matrimonio, etcétera) se representarán mediante los predicados binarios: *Padre, Hermano Político, Hermano, Hermana, Niño, Hija, Hijo, Esposo, Mujer, Marido, Abuelo, Nieto, Primo, Tía,* y *Tío*. Utilizaremos funciones para *Madre* y *Padre*, porque todas las personas tienen exactamente uno de cada uno (al menos de acuerdo con las reglas de la naturaleza).

Podemos pasar por cada función y predicado, apuntando lo que sabemos en términos de los otros símbolos. Por ejemplo, la madre de uno es uno de los padres y es femenino:

$$\forall x, y \;\; Madre(y) = x \Leftrightarrow Femenino(x) \wedge Padre(x, y).$$

El marido de uno es un esposo masculino:

$$\forall x, y \;\; Marido(y, x) \Leftrightarrow Masculino(y) \wedge Esposo(y, x).$$

Masculino y Femenino son categorías disjuntas:

$$\forall x \;\; Masculino(x) \Leftrightarrow \neg Femenino(x).$$

Padre e hijo son relaciones inversas:

$$\forall x, y \;\; Padre(x, y) \Leftrightarrow Hijo(y, x).$$

Un abuelo es el padre del padre de uno:

$$\forall x, y \;\; Abuelo(x, y) \Leftrightarrow \exists z \;\; Padre(x, z) \wedge Padre(z, y).$$

Un hermano es otro hijo del padre de uno:

$$\forall x, y \;\; Hermano(x, y) \Leftrightarrow x \neq y \wedge \exists z \;\; Padre(z, x) \wedge Padre(z, y).$$

Podríamos seguir con más páginas como esta, y el Ejercicio 8.11 pide que haga justamente eso.

AXIOMAS

Cada una de estas sentencias se puede ver como un **axioma** del dominio del parentesco. Los axiomas se asocian por lo general con dominios puramente matemáticos (veremos algunos axiomas sobre números en breve) pero la verdad es que se necesitan en todos los dominios. Los axiomas proporcionan la información factual esencial de la cual se pueden derivar conclusiones útiles. Nuestros axiomas de parentesco también son

DEFINICIONES

definiciones; tienen la forma $\forall x, y\ P(x, y) \Leftrightarrow \ldots$ Los axiomas definen la función *Madre* y los predicados *Marido*, *Masculino*, *Padre*, *Abuelo* y *Hermano* en términos de otros predicados. Nuestras definiciones «tocan el fondo» de un conjunto básico de predicados (*Hijo, Esposo,* y *Femenino*) sobre los cuales se definen los demás. Esta es una forma muy natural de desarrollar la representación de un dominio, y es análogo a la forma en que los paquetes de *software* se desarrollan a partir de definiciones sucesivas de subrutinas, partiendo de una biblioteca de funciones primitivas. Fíjese en que no hay necesariamente un único conjunto de predicados primitivos; podríamos perfectamente haber utilizado *Padre*, *Esposo* y *Masculino*. En algunos dominios, tal como veremos, no hay un conjunto básico claramente identificable.

TEOREMAS

No todas las sentencias lógicas acerca de un dominio son axiomas. Algunas son **teoremas**, es decir, son deducidas a partir de los axiomas. Por ejemplo, considere la aserción acerca de que la relación de hermandad es simétrica:

$$\forall x, y\ Hermano(x, y) \Leftrightarrow Hermano(y, x).$$

¿Es un axioma o un teorema? De hecho, es un teorema que lógicamente se sigue de los axiomas definidos para la relación de hermandad. Si PREGUNTAMOS a la base de conocimiento sobre esta sentencia, la base devolvería *verdadero*.

Desde un punto de vista puramente lógico, una base de conocimiento sólo necesita contener axiomas y no necesita contener teoremas, porque los teoremas no aumentan el conjunto de conclusiones que se siguen de la base de conocimiento. Desde un punto de vista práctico, los teoremas son esenciales para reducir el coste computacional para derivar sentencias nuevas. Sin ellos, un sistema de razonamiento tiene que empezar desde el principio cada vez, como si un físico tuviera que volver a deducir las reglas del cálculo con cada problema nuevo.

No todos los axiomas son definiciones. Algunos proporcionan información más general acerca de ciertos predicados sin tener que constituir una definición. Por el contrario, algunos predicados no tienen una definición completa porque no sabemos lo suficiente para caracterizarlos totalmente. Por ejemplo, no hay una manera obvia para completar la sentencia:

$$\forall x\ Persona(x) \Leftrightarrow \ldots$$

Afortunadamente, la lógica de primer orden nos permite hacer uso del predicado *Persona* sin definirlo completamente. En lugar de ello, podemos escribir especificaciones parciales de las propiedades que cada persona tiene y de las propiedades que hacen que algo sea una persona:

$$\forall x\ \ Persona(x) \Leftrightarrow \ldots$$
$$\forall x\ \ \ldots \Leftrightarrow Persona(x)$$

Los axiomas también pueden ser «tan sólo puros hechos», tal como *Masculino*(*Jaime*) y *Esposo*(*Jaime, Laura*). Este tipo de hechos forman las descripciones de las instancias de los problemas concretos, permitiendo así que se responda a preguntas concretas. Entonces, las respuestas a estas preguntas serán los teoremas que se siguen de los axiomas. A menudo, nos encontramos con que las respuestas esperadas no están disponibles, por ejemplo, de *Masculino*(*Jorge*) y *Esposo*(*Jorge, Laura*) esperamos ser capaces de in-

ferir *Femenino*(*Laura*); pero esta sentencia no se sigue de los axiomas dados anteriormente. Y esto es una señal de que nos hemos olvidado de algún axioma.

Números, conjuntos y listas

NÚMEROS NATURALES

AXIOMAS DE PEANO

Los números son quizás el ejemplo más gráfico de cómo se puede construir una gran teoría a partir de un núcleo de axiomas diminuto. Aquí describiremos la teoría de los **números naturales**, o la de los enteros no negativos. Necesitamos un predicado *NumNat* que será verdadero para los números naturales; necesitamos un símbolo de constante, 0; y necesitamos un símbolo de función, S (sucesor). Los **axiomas de Peano** definen los números naturales y la suma[8]. Los números naturales se definen recursivamente:

$$NumNat(0)$$
$$\forall n\ NumNat(n) \Rightarrow NumNat(S(n))$$

Es decir, 0 es un número natural, y para cada objeto n, si n es un número natural entonces el $S(n)$ es un número natural. Así, los números naturales son el 0, el $S(0)$, el $S(S(0))$, etcétera. También necesitamos un axioma para restringir la función sucesor:

$$\forall n\ 0 \neq S(n)$$
$$\forall m, n\ m \neq n \Rightarrow S(m)\ m\ S(n)$$

Ahora podemos definir la adición (suma) en términos de la función sucesor:

$$\forall m\ NumNat(m) \Rightarrow +(m, 0) = m$$
$$\forall m, n\ NumNat(m) \wedge NumNat(n) \Rightarrow +(S(m), n) = S(+(m, n))$$

INFIJA

PREFIJA

El primero de estos axiomas dice que sumar 0 a cualquier número natural m da el mismo m. Fíjese en el uso de la función binaria «+» en el término $+(m, 0)$; en las matemáticas habituales el término estaría escrito $m + 0$, utilizando la notación **infija**. (La notación que hemos utilizado para la lógica de primer orden se denomina **prefija**.) Para hacer que nuestras sentencias acerca de los números sean más fáciles de leer permitiremos el uso de la notación infija. También podemos escribir $S(n)$ como $n + 1$, entonces el segundo axioma se convierte en

$$\forall m, n\ NumNat(m) \wedge NumNat(n) \Rightarrow (m + 1) + n = (m + n) + 1$$

Este axioma reduce la suma a la aplicación repetida de la función sucesor.

SINTAXIS EDULCORADA

El uso de la notación infija es un ejemplo de **sintaxis edulcorada**, es decir, una ampliación o abreviación de una sintaxis estándar que no cambia su semántica. Cualquier sentencia que está edulcorada puede «des-edulcorarse» para producir una sentencia equivalente en la habitual lógica de primer orden.

Una vez tenemos la suma, es fácil definir la multiplicación como una suma repetida, la exponenciación como una multiplicación repetida, la división entera y el resto, los

[8] Los axiomas de Peano también incluyen el principio de inducción, que es una sentencia de lógica segundo orden más que de lógica de primer orden. La importancia de esta diferencia se explica en el Capítulo 9.

números primos, etcétera. De este modo, la totalidad de la teoría de los números (incluyendo la criptografía) se puede desarrollar a partir de una constante, una función, un predicado y cuatro axiomas.

CONJUNTOS

El dominio de los **conjuntos** es tan fundamental para las matemáticas como para el razonamiento del sentido común. (De hecho, es posible desarrollar la teoría de los números con base en la teoría de los conjuntos.) Queremos ser capaces de representar conjuntos individuales, incluyendo el conjunto vacío. Necesitamos un mecanismo para construir conjuntos añadiendo un elemento a un conjunto o tomando la unión o la intersección de dos conjuntos. Querremos saber si un elemento es un miembro de un conjunto, y ser capaces de distinguir conjuntos de objetos que no son conjuntos.

Utilizaremos el vocabulario habitual de la teoría de conjuntos como sintaxis edulcorada. El conjunto vacío es una constante escrita como $\{\,\}$. Hay un predicado unitario, *Conjunto*, que es verdadero para los conjuntos. Los predicados binarios son $x \in s$ (x es un miembro del conjunto s) y $s_1 \subseteq s_2$ (el conjunto s_1 es un subconjunto, no necesariamente propio, del conjunto s_2). Las funciones binarias son $s_1 \cap s_2$ (la intersección de dos conjuntos), $s_1 \cup s_2$ (la unión de dos conjuntos), y $\{x|s\}$ (el conjunto resultante de añadir el elemento x al conjunto s). Un conjunto posible de axiomas es el siguiente:

1. Los únicos conjuntos son el conjunto vacío y aquellos construidos añadiendo algo a un conjunto:

$$\forall s \;\; Conjunto(s) \;\Leftrightarrow\; (s = \{\,\}) \vee (\exists x, s_2 \;\; Conjunto(s_2) \wedge s = \{x|s_2\})$$

2. El conjunto vacío no tiene elementos añadidos a él, en otras palabras, no hay forma de descomponer un *ConjuntoVacío* en un conjunto más pequeño y un elemento:

$$\neg\exists x, s \;\; \{x|s\} = \{\,\}$$

3. Añadir un elemento que ya pertenece a un conjunto no tiene ningún efecto:

$$\forall x, s \;\; x \in s \;\Leftrightarrow\; s = \{x|s\}$$

4. Los únicos elementos de un conjunto son los que fueron añadidos a él. Expresamos esto recursivamente, diciendo que x es un miembro de s si y sólo si s es igual a algún conjunto s_2 al que se le ha añadido un elemento y, y que y era el mismo elemento que x o que x es un miembro de s_2:

$$\forall x, s \;\; x \in s \;\Leftrightarrow\; [\exists y, s_2 \;\; (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$$

5. Un conjunto es un subconjunto de otro conjunto si y sólo si todos los miembros del primer conjunto son miembros del segundo conjunto:

$$\forall s_1, s_2 \;\; s_1 \subseteq s_2 \;\Leftrightarrow\; (\forall x \;\; x \in s_1 \;\Rightarrow\; x \in s_2)$$

6. Dos conjuntos son iguales si y sólo si cada uno es subconjunto del otro:

$$\forall s_1, s_2 \;\; (s_1 = s_2) \;\Leftrightarrow\; (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

7. Un objeto pertenece a la intersección de dos conjuntos si y sólo si es miembro de ambos conjuntos:

$$\forall x, s_1, s_2 \;\; x \in (s_1 \cap s_2) \;\Leftrightarrow\; (x \in s_1 \wedge x \in s_2)$$

8. Un objeto pertenece a la unión de dos conjuntos si y sólo si es miembro de alguno de los dos:

$$\forall x, s_1, s_2 \;\; x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$$

LISTAS

Las **listas** son muy parecidas a los conjuntos. Las diferencias son que las listas están ordenadas y que el mismo elemento puede aparecer más de una vez en una lista. Podemos utilizar el vocabulario del Lisp para las listas: *Nil* es la constante para las listas sin elementos; *Cons, Unir, Primero* y *Resto* son funciones; y *Encontrar* es el predicado que hace en listas lo que *Miembro* hace en conjuntos. *¿Lista?* es un predicado que es verdadero sólo para las listas. Como en los conjuntos, es común el uso de sintaxis edulcoradas en las sentencias lógicas que tratan sobre listas. La lista vacía es [ ]. El término *Cons*(x, y), donde y es un conjunto no vacío, se escribe [$x|y$]. El término *Cons*(x, *Nil*), (por ejemplo, la lista conteniendo el elemento x), se escribe [x]. Una lista con varios elementos, como [*A, B, C*], se corresponde al término anidado *Cons*(*A*, *Cons*(*B*, *Cons*(*C*, *Nil*))). El Ejercicio 8.14 pide que escriba los axiomas para las listas.

El mundo de *wumpus*

En el Capítulo 7 se dieron algunos axiomas para el mundo de *wumpus* en lógica proposicional. Los axiomas en lógica de primer orden de esta sección son mucho más precisos, capturando de forma natural exactamente lo que queremos expresar.

Recuerde que el agente *wumpus* recibe un vector de percepciones con cinco elementos. La sentencia en primer orden correspondiente almacenada en la base de conocimiento debe incluir tanto la percepción como el instante de tiempo en el que ocurrió ésta, de otra manera el agente se confundiría acerca de cuándo vio qué cosa. Utilizaremos enteros para los instantes de tiempo. Una típica sentencia de percepción sería

Percepción([*MalHedor, Brisa, Resplandor, Nada, Nada*], 5)

Aquí, *Percepción* es un predicado binario y *MalHedor* y otros son constantes colocadas en la lista. Las acciones en el mundo de *wumpus* se pueden representar mediante términos lógicos:

Girar(*Derecha*), *Girar*(*Izquierda*), *Avanzar, Disparar, Agarrar, Libertar, Escalar*

Para hallar qué acción es la mejor, el programa del agente construye una petición como esta

$$\exists a \;\; MejorAcción(a, 5)$$

PREGUNTAR resolvería esta petición y devolvería una lista de ligaduras como {*a*/*Agarrar*}. El programa del agente entonces puede devolver *Agarrar* como la acción que debe llevar a cabo, pero primero debe DECIR a la propia base de conocimiento que está ejecutando la acción *Agarrar*.

Los datos acerca de las crudas percepciones implican ciertos hechos acerca del estado actual. Por ejemplo:

$$\forall t, s, g, m, c \; Percepción([s, Brisa, g, m, c], t) \Rightarrow Brisa(t),$$
$$\forall t, s, b, m, c \; Percepción([s, b, Resplandor, m, c], t) \Rightarrow Resplandor(t),$$

etcétera. Estas reglas muestran una forma trivial del proceso de razonamiento denominado **percepción**, que estudiaremos en profundidad en el Capítulo 24. Fíjese en la cuantificación sobre *t*. En la lógica proposicional, habríamos necesitado copias de cada sentencia para cada instante de tiempo.

El comportamiento simple de tipo «reflexivo» también puede ser implementado mediante sentencias de implicación cuantificada. Por ejemplo, tenemos

$$\forall t\ Resplandor(t) \Rightarrow MejorAcción(Agarrar, t)$$

Dadas las percepciones y las reglas de los párrafos precedentes, esto nos daría la conclusión deseada *MejorAcción*(*Agarrar*, 5) (es decir, Agarrar es lo más correcto a hacer). Fíjese en la correspondencia entre esta regla y la conexión directa percepción-acción de los agentes basados en circuitos de la Figura 7.20; la conexión en el circuito se cuantifica *implícitamente* sobre el tiempo.

SINCRÓNICA

DIACRÓNICA

Hasta ahora en esta sección, las sentencias que tratan con el tiempo han sido sentencias **sincrónicas** («al mismo tiempo»), es decir, las sentencias relacionan propiedades del estado del mundo con otras propiedades del mismo estado del mundo. Las sentencias que permiten razonar «a través del tiempo» se denominan **diacrónicas**; por ejemplo, el agente necesita saber combinar la información acerca de sus localizaciones anteriores con la información acerca de la acción que acaba de realizar, para establecer su localización actual. Aplazaremos la discusión acerca de las sentencias diacrónicas hasta el Capítulo 10; por ahora, sólo asuma que las inferencias necesarias se han realizado para los predicados de localización, y otros, dependientes del tiempo.

Hemos representado las percepciones y las acciones; ahora es el momento de representar el propio entorno. Vamos a empezar con los objetos. Los candidatos obvios son las casillas, los hoyos y el *wumpus*. Podríamos nombrar cada casilla ($Casilla_{1,2}$, etcétera) pero entonces el hecho de que la $Casilla_{1,2}$ y la $Casilla_{1,3}$ estén adyacentes tendría que ser un hecho «extra», y necesitaríamos un hecho de este tipo para cada par de casillas. Es mejor utilizar un término complejo en el que la fila y la columna aparezcan como enteros; por ejemplo, simplemente podemos usar la lista de términos [1, 2]. La adyacencia entre dos casillas se puede definir mediante

$$\forall x, y, a, b,\ Adyacente([x, y], [a, b]) \Leftrightarrow$$
$$[a, b] \in \{[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]\}$$

También podríamos nombrar cada hoyo, pero sería inapropiado por otro motivo: no hay ninguna razón para distinguir a unos hoyos de otros[9]. Es mucho más sencillo utilizar un predicado unario *Hoyo* que es verdadero en las casillas que contengan hoyos. Por último, como sólo hay exactamente un *wumpus*, una constante *Wumpus* es tan buena como un predicado unitario (y quizá más digno para el punto de vista del *wumpus*). El *wumpus* vive exactamente en una casilla, por tanto es una buena idea utilizar una función como *Casa*(*Wumpus*) para nombrar la casilla. Esto evita por completo el enorme conjunto de

[9] De forma similar, muchos de nosotros no nombramos cada pájaro que vuela sobre nuestras cabezas en sus migraciones a regiones más cálidas en invierno. Un ornitólogo que desea estudiar los patrones de migración, los ratios de supervivencia, etcétera, nombraría cada pájaro por medio de una alarma en su pata, porque cada pájaro debe ser observado.

sentencias que se necesitaban en la lógica proposicional para decir que una casilla en concreto contenía al *wumpus*. (Aún sería mucho peor para la lógica proposicional si hubieran dos *wumpus*.)

La localización del agente cambia con el tiempo, entonces escribiremos $En(Agente, s, t)$ para indicar que el agente se encuentra en la casilla s en el instante t. Dada su localización actual, el agente puede inferir las propiedades de la casilla a partir de las propiedades de su percepción actual. Por ejemplo, si el agente se encuentra en una casilla y percibe una brisa, entonces la casilla tiene una corriente de aire:

$$\forall s, t \;\; En(Agente, s, t) \wedge Brisa(t) \Rightarrow CorrienteAire(s)$$

Es útil saber si una *casilla* tiene una corriente de aire porque sabemos que los hoyos no pueden desplazarse. Fíjese en que *CorrienteAire* no tiene el argumento del tiempo.

Habiendo descubierto qué casillas tienen brisa (o son apestosas) y, muy importante, las que *no* tienen brisa (o *no* son apestosas), el agente puede deducir dónde están los hoyos (y dónde está el *wumpus*). Hay dos tipos de reglas sincrónicas que podrían permitir sacar este tipo de deducciones:

REGLAS DE DIAGNÓSTICO

- **Reglas de diagnóstico:**
  Las reglas de diagnóstico nos llevan de los efectos observados a sus causas ocultas. Para encontrar hoyos, las reglas obvias de diagnóstico dicen que si una casilla tiene una brisa, alguna casilla adyacente debe contener un hoyo, o

  $$\forall s \; CorrienteAire(s) \Rightarrow \exists r \; Adyacente(r, s) \wedge Hoyo(r)$$

  y si una casilla no tiene una brisa, ninguna casilla adyacente contiene un hoyo[10]:

  $$\forall s \; \neg CorrienteAire(s) \Rightarrow \neg\exists r \; Adyacente(r, s) \wedge Hoyo(r)$$

  Combinando estas dos reglas, obtenemos la siguiente sentencia bicondicional

  $$\forall s \; CorrienteAire(s) \Leftrightarrow \exists r \; Adyacente(r, s) \wedge Hoyo(r) \qquad (8.3)$$

REGLAS CAUSALES

- **Reglas causales:**
  Las reglas causales reflejan la dirección que se asume de causalidad en el mundo: algunas propiedades ocultas del mundo causan que se generen ciertas percepciones. Por ejemplo, un hoyo causa que todas sus casillas adyacentes tengan una brisa:

  $$\forall r \; Hoyo(r) \Rightarrow [\forall s \; Adyacente(s, r) \Rightarrow CorrienteAire(s)]$$

  y si todas las casillas adyacentes a una casilla dada no tienen hoyos, la casilla no tiene brisa:

  $$\forall s \; [\forall r \; Adyacente(r, s) \Rightarrow \neg Hoyo(r)] \Rightarrow \neg CorrienteAire(s)$$

[10] Hay una tendencia humana natural en olvidar anotar la información negativa de este tipo. En una conversación esta tendencia es totalmente normal (sería muy extraño decir «Hay dos copas en la mesa *y no hay tres o más*,» aunque pensar que «Hay dos copas en la mesa» sigue siendo verdadero, estrictamente hablando, cuando hay tres o más). Retomaremos este tema en el Capítulo 10.

Con algo de esfuerzo, es posible demostrar que estas dos sentencias juntas son equivalentes lógicamente a la sentencia bicondicional de la Ecuación (8.3). También se puede pensar en la propia bicondicional como en una regla causal, porque describe cómo se genera el valor de verdad de *CorrienteAire* a partir del estado del mundo.

RAZONAMIENTO BASADO EN MODELOS

Los sistemas que razonan con reglas causales se denominan sistemas de **razonamiento basado en modelos**, porque las reglas causales forman un modelo de cómo se comporta el entorno. La diferencia entre el razonamiento basado en modelos y el de diagnóstico es muy importante en muchas áreas de la IA. El diagnóstico médico en concreto ha sido un área de investigación muy activa, en la que los enfoques basados en asociaciones directas entre los síntomas y las enfermedades (un enfoque de diagnóstico) han sido reemplazados gradualmente por enfoques que utilizan un modelo explícito del proceso de la enfermedad y de cómo se manifiesta en los síntomas. Estos temas se presentarán también en el Capítulo 13.

Cualquier tipo de representación que el agente utilice, *si los axiomas describen correcta y completamente la forma en que el mundo se comporta y la forma en que las percepciones se producen, entonces cualquier procedimiento de inferencia lógica completo inferirá la posible descripción del estado del mundo más robusta, dadas las percepciones disponibles*. Así que el diseñador del agente puede concentrarse en obtener el conocimiento acorde, sin preocuparse demasiado acerca del proceso de deducción. Además, hemos visto que la lógica de primer orden puede representar el mundo de *wumpus*, y no de forma menos precisa que la descripción en lenguaje natural dada en el Capítulo 7.

8.4 Ingeniería del conocimiento con lógica de primer orden

INGENIERÍA DEL CONOCIMIENTO

La sección anterior ilustraba el uso de la lógica de primer orden para representar el conocimiento de tres dominios sencillos. Esta sección describe el proceso general de construcción de una base de conocimiento (un proceso denominado **ingeniería del conocimiento**). Un ingeniero del conocimiento es alguien que investiga un dominio concreto, aprende qué conceptos son los importantes en ese dominio, y crea una representación formal de los objetos y relaciones del dominio. Ilustraremos el proceso de ingeniería del conocimiento en un dominio de circuitos electrónicos, que imaginamos ya es bastante familiar, para que nos podamos concentrar en los temas representacionales involucrados. El enfoque que tomaremos es adecuado para desarrollar bases de conocimiento de *propósito-específico* cuyo dominio se circunscribe cuidadosamente y cuyo rango de peticiones se conoce de antemano. Las bases de conocimiento de *propósito-general*, cuya intención es que apoyen las peticiones de todo el abanico del conocimiento humano, se discutirán en el Capítulo 10.

El proceso de ingeniería del conocimiento

Los proyectos de ingeniería del conocimiento varían ampliamente en su contenido, alcance, y dificultad, pero todos estos proyectos incluyen los siguientes pasos:

1. *Identificar la tarea*. El ingeniero del conocimiento debe delinear el rango de las preguntas que la base de conocimiento debe soportar y los tipos de hechos que estarán disponibles para cada instancia de problema en particular. Por ejemplo, ¿la base de conocimiento *wumpus* necesita ser capaz de escoger las acciones o se requiere que sólo responda a las preguntas acerca del contenido del entorno? ¿Los hechos sensoriales incluirán la localización actual? La tarea determinará qué conocimiento debe ser representado para conectar las instancias de los problemas a las respuestas. Este paso es análogo al proceso REAS del diseño de agentes del Capítulo 2.
2. *Recopilar el conocimiento relevante*. El ingeniero del conocimiento debería ser ya un experto en el dominio, o debería necesitar trabajar con expertos reales para extraer el conocimiento que ellos poseen (en un proceso denominado **adquisición del conocimiento**). En esta fase el conocimiento no se representa formalmente. La idea es entender el alcance de la base de conocimiento, tal como se determinó en la tarea, y entender cómo trabaja realmente el dominio.

   ADQUISICIÓN DEL CONOCIMIENTO

   Para el ejemplo del mundo de *wumpus*, que está definido por un conjunto artificial de reglas, el conocimiento relevante es fácil de identificar. (Sin embargo, fíjese que la definición de adyacencia no estaba suministrada explícitamente por las reglas del mundo de *wumpus*.) Para los dominios reales, el tema de la relevancia puede ser bastante difícil, por ejemplo, un sistema de simulación de diseños de VLSI podría, o no, necesitar tener en cuenta las pérdidas de capacitación o los efectos de revestimiento.
3. *Decidir el vocabulario de los predicados, funciones y constantes*. Es decir, traducir los conceptos importantes del nivel del dominio a nombres del nivel lógico. Esto involucra a muchas cuestiones de *estilo* de la ingeniería del conocimiento. Al igual que el estilo de la programación, esto puede tener un impacto significativo en el éxito final del proyecto. Por ejemplo, ¿los hoyos estarían representados por objetos o por un predicado unitario aplicado a las casillas? ¿La orientación del agente estaría representada por una función o un predicado? ¿La localización del *wumpus* dependería del tiempo? Una vez se han realizado las elecciones, el resultado es un vocabulario que se conoce por la **ontología** del dominio. La palabra *ontología* indica una teoría concreta sobre la naturaleza del ser o de la existencia de algo. La ontología establece qué tipo de cosas existen, pero no determina sus propiedades e interrelaciones específicas.

   ONTOLOGÍA
4. *Codificar el conocimiento general acerca del dominio*. El ingeniero del conocimiento anota los axiomas para todos los términos del vocabulario. Esto hace que se defina (en todo lo posible) el significado de los términos, permitiendo al experto comprobar el contenido. A menudo, esta fase revela ideas equivocadas o lagunas en el vocabulario, que deberán fijarse volviendo al paso 3 y repitiendo los pasos del proceso.

5. *Codificar una descripción de la instancia de un problema específico*. Si la ontología está bien pensada este paso será fácil. Consistirá en escribir sentencias atómicas sencillas acerca de instancias de conceptos que ya son parte de la ontología. Para un agente lógico, las instancias del problema se obtienen de los sensores, mientras que para una base de conocimiento «no corpórea» se obtienen de sentencias adicionales de la misma manera que un programa tradicional las obtiene de los datos de entrada.
6. *Plantear peticiones al procedimiento de inferencia y obtener respuestas*. Esta es la fase en donde se obtiene la recompensa: podemos dejar al procedimiento de inferencia trabajar sobre los axiomas y los hechos del problema concreto para derivar los hechos que estamos interesados en conocer.
7. *Depurar la base de conocimiento*. Rara vez, las respuestas a las peticiones son correctas en un primer intento. Más concretamente, las respuestas serán correctas *para la base de conocimiento como si fueran escritas*, asumiendo que el procedimiento de inferencia sea sólido, pero no serán las que el usuario estaba esperando. Por ejemplo, si falta un axioma, algunas peticiones no serán respondidas por la base de conocimiento. Esto podría resultar un proceso de depuración considerable. Axiomas ausentes o axiomas que son demasiado débiles se pueden identificar fácilmente fijándonos en los sitios donde la cadena del razonamiento para inesperadamente. Por ejemplo, si la base de conocimiento incluye uno de los axiomas de diagnóstico para hoyos,

$$\forall s\ CorrienteAire(s) \Rightarrow \exists r\ Adyacente(r, s) \wedge Hoyo(r)$$

pero no el otro axioma, entonces el agente nunca será capaz de demostrar la *ausencia* de hoyos. Los axiomas incorrectos se pueden identificar porque son enunciados falsos acerca del mundo. Por ejemplo, la sentencia

$$\forall x\ NumPatas(x, 4) \Rightarrow Mamífero(x)$$

es falsa para los reptiles, anfibios, y mucho más importante, para las mesas. *La falsedad de esta sentencia se puede determinar independientemente del resto de la base de conocimiento*. En contraste, un error típico en un programa se parece a éste:

```
offset[11] = posición + 1
```

Es imposible decir si este enunciado es correcto sin mirar el resto del programa para ver si, por ejemplo, *offset* se utiliza para referirse a la posición actual, o a una más allá de la posición actual, o si el valor de posición es cambiado por otro enunciado y así *offset* sería otra vez cambiado.

Para entender mejor este proceso de siete pasos vamos a aplicarlo a un ejemplo ampliado: el dominio de los circuitos electrónicos.

El dominio de los circuitos electrónicos

Desarrollaremos una ontología y una base de conocimiento que nos permitirá razonar acerca de los circuitos digitales del tipo como el que se muestra en la Figura 8.4. Seguiremos el proceso de los siete pasos de la ingeniería del conocimiento.

[11] En español *compensación*, aunque en tipografía se utiliza el término *offset*. (N del RT.)

Figura 8.4 El circuito digital C1, pretendiendo que sea un sumador de un bit. Las dos primeras entradas son los dos bits a ser sumados y la tercera entrada es un bit de acarreo. La primera salida es la suma, y la segunda salida es el bit de acarreo para la siguiente adición. El circuito contiene dos puertas XOR, dos puertas AND y una puerta OR.

Identificar la tarea

Hay muchas tareas de razonamiento relacionadas con los circuitos digitales. En el nivel más alto, uno analiza la funcionalidad del circuito. Por ejemplo, ¿el circuito de la Figura 8.4 realmente suma correctamente? Si todas las entradas son buenas, ¿cuál es la salida de la puerta A2? También son interesantes las preguntas acerca de la estructura del circuito. Por ejemplo, ¿están todas las puertas conectadas a la primera terminal de entrada? ¿El circuito contiene bucles de retroalimentación? Estas serán nuestras tareas en esta sección. Hay más niveles de análisis detallados, incluyendo aquellos relacionados con retardos temporales, superficie del circuito, consumo de corriente, coste de producción, etcétera. Cada uno de estos niveles requeriría de un conocimiento adicional.

Recopilar el conocimiento relevante

¿Qué sabemos acerca de los circuitos digitales? Para nuestros propósitos, los circuitos digitales están compuestos por cables y puertas. La señal circula por los cables a las terminales entrada de las puertas, y cada puerta produce una señal en la terminal salida que circula a través de otro cable. Para determinar cómo serán estas señales, necesitamos saber cómo transforman las puertas su señal de entrada. Hay cuatro tipos de puertas: las puertas AND, OR y XOR tienen dos terminales de entrada, las puertas NOT tienen una. Todas las puertas tienen una terminal de salida. Los circuitos, al igual que las puertas, tienen terminales de entrada y de salida.

Para razonar acerca de la funcionalidad y la conectividad no necesitamos hablar acerca de los propios cables, los recorridos que los cables realizan, o los empalmes en los que dos cables se juntan. Todo lo que nos importa son las conexiones entre los terminales (podemos decir que un terminal de salida está conectado con otro terminal de entrada sin tener que mencionar el cable que realmente los conecta). Hay muchos factores

del dominio que son irrelevantes para nuestro análisis, tales como el tamaño, la forma, el color, o el coste de los diferentes componentes del circuito.

Si nuestro propósito fuera algo distinto a la verificación del diseño al nivel de puertas, la ontología sería diferente. Por ejemplo, si estuviéramos interesados en depurar circuitos defectuosos, entonces probablemente sería una buena idea incluir los cables en la ontología, porque un cable defectuoso puede corromper el flujo de señal que pasa por él. Para resolver defectos de tiempo, necesitaríamos incluir puertas de retardo. Si estuviéramos interesados en diseñar un producto que fuera rentable, entonces serían importantes el coste del circuito y su velocidad relativa a otros productos del mercado.

Decidir el vocabulario

Ahora sabemos que queremos hablar acerca de circuitos, terminales, señales y puertas. El siguiente paso es elegir las funciones, predicados y constantes para representarlos. Comenzaremos a partir de las puertas individuales y ascenderemos a los circuitos.

Lo primero, necesitamos ser capaces de distinguir una puerta de las otras. Esto se controla nombrando las puertas con constantes: X_1, X_2, etcétera. Aunque cada puerta está conectada en el circuito en su manera particular, su *comportamiento* (la forma en que transforma las señales de entrada en la señal de salida) sólo depende de su *tipo*. Podemos utilizar una función para referirnos al tipo de puerta[12]. Por ejemplo, podemos escribir $Tipo(X_1) = XOR$. Esto introduce la constante *XOR* para un tipo concreto de puerta; las otras constantes se llamarán *OR, AND* y *NOT*, la función *Tipo* no es la única forma de codificar la distinción ontológica. Podríamos haber utilizado un predicado binario, $Tipo(X_1, XOR)$, o diferentes predicados individuales, como $XOR(X_1)$. Cualquiera de estas elecciones trabajaría correctamente, pero al elegir la función *Tipo* evitamos la necesidad de un axioma que diga que cada puerta individual sólo puede ser de un tipo. La semántica de la función ya lo garantiza.

Ahora consideraremos los terminales. Una puerta o un circuito puede tener uno o más terminales de entrada y uno o más terminales de salida. Simplemente podríamos nombrar cada uno con una constante, igual que hicimos con las puertas. De esta manera, la puerta X_1 podría tener los terminales cuyos nombres serían $X_1Entrada_1$, $X_1Entrada_2$, y $X_1Salida_1$. Sin embargo, la tendencia a generar nombres compuestos largos se debería evitar. Llamar a algo $X_1Entrada_1$ no hace que sea la primera entrada de X_1; necesitaríamos decir esto utilizando una aserción explícita. Probablemente es mejor nombrar la puerta utilizando una función, tal como nombramos la pierna izquierda del Rey Juan con *PiernaIzquierda*(*Juan*). Entonces dejemos que $Entrada(1, X_1)$ denote el terminal de la primera entrada de la puerta X_1. Utilizaríamos una función similar *Salida* para los terminales de salida.

La conectividad entre las puertas se puede representar por el predicado *Conectado*, que toma dos terminales como argumentos, como en $Conectado(Salida(1, X_1), Entrada(1, X_2))$.

[12] Fíjese en que hemos utilizado nombres con las letras adecuadas (A_1, X_1, etcétera) simplemente para hacer más fácil la lectura del ejemplo. La base de conocimiento también debe contener información sobre los tipos de puertas.

Por último, necesitamos saber si una señal está *on* u *off*. Una posibilidad es utilizar un predicado unitario, *On*, que sea verdadero cuando la señal de un terminal es *on*. Sin embargo, esto hace un poco difícil plantear preguntas tales como «¿Cuáles son todos los posibles valores de las señales de los terminales de salida del circuito C_1?» Por lo tanto, introduciremos los dos «valores de la señal» como las constantes 1 y 0, y una función *Señal* que tome un terminal como argumento y denote el valor de señal para ese terminal.

Codificar el conocimiento general del dominio

Un síntoma de que poseemos una buena ontología es que haya pocas reglas generales que se necesiten especificar. Un síntoma de que tengamos un buen vocabulario es que cada regla se pueda representar de forma clara y precisa. En nuestro ejemplo, sólo necesitamos siete reglas sencillas para describir cada cosa que necesitamos saber acerca de los circuitos:

1. Si dos terminales están conectados entonces tienen la misma señal:

$$\forall t_1, t_2 \;\; Conectado(t_1, t_2) \Rightarrow Señal(t_1) = Señal(t_2)$$

2. La señal de cada terminal es o bien 1 o bien 0 (pero no ambos):

$$\forall t \;\; Señal(t) = 1 \vee Señal(t) = 0$$
$$1 \neq 0$$

3. El predicado Conectado es conmutativo:

$$\forall t_1, t_2 \;\; Conectado(t_1, t_2) \Leftrightarrow Conectado(t_2, t_1)$$

4. La salida de una puerta OR es 1 si y sólo si alguna de sus entradas son 1:

$$\forall g \;\; Tipo(g) = OR \Rightarrow Señal(Salida(1, g)) = 1 \Leftrightarrow \exists n \;\; Señal(Entrada(n, g)) = 1$$

5. La salida de una puerta AND es 0 si y sólo si cualquiera de sus entradas es 0:

$$\forall g \;\; Tipo(g) = AND \Rightarrow Señal(Salida(1, g)) = 0 \Leftrightarrow \exists n \;\; Señal(Entrada(n, g)) = 0$$

6. La salida de una puerta XOR es 1 si y sólo si sus entradas son diferentes:

$$\forall g \;\; Tipo(g) = XOR \Rightarrow$$
$$Señal(Salida(1, g)) = 1 \Leftrightarrow Señal(Entrada(1, g)) \neq Señal(Entrada(2, g))$$

7. La salida de una puerta NOT es la opuesta de su entrada:

$$\forall g \;\; (Tipo(g) = NOT) \Rightarrow Señal(Salida(1, g)) \neq Señal(Entrada(1, g))$$

Codificar la instancia del problema específico

El circuito que se muestra en la Figura 8.4 está codificado como el circuito C_1, con la siguiente descripción. Lo primero que hacemos es categorizar las puertas:

$$Tipo(X_1) = XOR \qquad Tipo(X_2) = XOR$$
$$Tipo(A_1) = AND \qquad Tipo(A_2) = AND$$
$$Tipo(O_1) = OR$$

Entonces, mostramos las conexiones entre ellas:

$Conectado(Salida(1, X_1), Entrada(1, X_2))$ $Conectado(Entrada(1, C_1), Entrada(1, X_1))$
$Conectado(Salida(1, X_1), Entrada(2, A_2))$ $Conectado(Entrada(1, C_1), Entrada(1, A_1))$
$Conectado(Salida(1, A_2), Entrada(1, O_1))$ $Conectado(Entrada(2, C_1), Entrada(2, X_1))$
$Conectado(Salida(1, A_1), Entrada(2, O_1))$ $Conectado(Entrada(2, C_1), Entrada(2, A_1))$
$Conectado(Salida(1, X_2), Salida\ (1, C_1))$ $Conectado(Entrada(3, C_1), Entrada(2, X_2))$
$Conectado(Salida(1, O_1), Salida\ (2, C_1))$ $Conectado(Entrada(3, C_1), Entrada(1, A_2))$

Plantear peticiones al procedimiento de inferencia

¿Qué combinación de entradas causaría que la primera salida de C_1 (la suma de bits) sea 0 y que la segunda salida de C_1 (el bit de acarreo) sea 1?

$$\exists i_1, i_2, i_3\ \ Señal(Entrada(1, C_1)) = i_1 \wedge Señal(Entrada(2, C_1)) = i_2$$
$$\wedge\ Señal(Entrada(3, C_1)) = i_3 \wedge Señal(Salida(1, C_1)) = 0 \wedge Señal(Salida(2, C_1)) = 1$$

Las respuestas serían las sustituciones de las variables i_1, i_2, e i_3 de tal modo que la sentencia resultante esté implicada de la base de conocimiento. Hay tres posibles sustituciones:

$$\{i_1/1, i_2/1, i_3/0\} \quad \{i_1/1, i_2/0, i_3/1\} \quad \{i_1/0, i_2/1, i_3/1\}$$

¿Cuáles son los posibles conjuntos de valores de los terminales para el circuito de suma?

$$\exists i_1, i_2, i_3, o_1, o_2\ \ Señal(Entrada(1, C_1)) = i_1 \wedge Señal(Entrada(2, C_1)) = i_2$$
$$\wedge\ Señal(Entrada(3, C_1)) = i_3 \wedge Señal(Salida(1, C_1)) = o_1 \wedge Señal(Salida(2, C_1)) = o_2$$

VERIFICACIÓN DE CIRCUITOS

Esta última petición devuelve una tabla completa de entradas-salidas para el dispositivo, que se puede utilizar para comprobar si en efecto realiza correctamente la suma. Este es un ejemplo sencillo de la **verificación de circuitos**. También podemos utilizar la definición del circuito para construir sistemas digitales más grandes, sobre los cuales, se puede aplicar el mismo tipo de procedimiento de verificación. (*Véase* Ejercicio 8.17.) Muchos dominios son tratables con el mismo método de desarrollo de una base de conocimiento estructurada, en los que los conceptos más complejos se definen con base en los más sencillos.

Depurar la base de conocimiento

Podemos perturbar la base de conocimiento de muchas maneras para averiguar qué tipos de comportamientos erróneos emergen. Por ejemplo, suponga que omitimos la aserción acerca de que $1 \neq 0$[13]. De repente, el sistema será incapaz de demostrar las salidas del circuito, excepto en los casos en los que las entradas sean 000 y 110. Podemos lo-

[13] Este tipo de omisión es bastante común porque los humanos típicamente asumen que nombres diferentes se refieren a cosas diferentes. Los sistemas de programación lógica, que se describen en el Capítulo 9, también realizan esta asunción.

calizar el problema preguntando por las salidas de cada puerta. Por ejemplo, podemos preguntar

$$\exists i_1, i_2, o \;\; Señal(Entrada(1, C_1)) = i_1 \wedge Señal(Entrada(2, C_1)) = i_2 \wedge Señal(Salida(1, X_1))$$

que revela que no se conocen las salidas de la puerta X_1 para los casos en los que las entradas son 10 y 01. Entonces, observamos el axioma acerca de las puertas XOR, aplicada a la puerta X_1:

$$Señal(Salida(1, X_1)) = 1 \;\Leftrightarrow\; Señal(Entrada(1, X_1)) \neq Señal(Entrada(2, X_1))$$

Si se sabe que las entradas deben ser 1 y 0, entonces esto se reduce a

$$Señal(Salida(1, X_1)) = 1 \;\Leftrightarrow\; 1 \neq 0$$

Ahora el problema es aparente: el sistema es incapaz de inferir que *Señal*(*Salida*(1, X1)) = 1, por lo tanto, necesitamos decirle que $1 \neq 0$.

8.5 Resumen

En este capítulo hemos introducido la **lógica de primer orden**, un lenguaje de representación que es mucho más potente que la lógica proposicional. Los puntos importantes son los siguientes:

- Los lenguajes de representación del conocimiento deberían ser declarativos, composicionales, expresivos, independientes del contexto, y no ambiguos.
- Las lógicas difieren en sus **compromisos ontológicos** y **compromisos epistemológicos**. Mientras que la lógica proposicional se compromete sólo con la existencia de hechos, la lógica de primer orden se compromete con la existencia de objetos y sus relaciones, y por ello gana poder expresivo.
- Un **mundo posible**, o **modelo**, se define para la lógica de primer orden como un conjunto de objetos, las relaciones entre ellos y las funciones que se les puede aplicar.
- Los **símbolos de constante** identifican los objetos, los **símbolos de predicado** identifican las relaciones, y los **símbolos de función** identifican las funciones. Una interpretación especifica una aplicación de los símbolos al modelo. Los **términos complejos** aplican símbolos de función a los términos para identificar un objeto. Dados un modelo y una interpretación, se determina el valor de verdad de la sentencia.
- Una **sentencia atómica** consiste en un predicado aplicado a uno o más términos; el predicado es verdadero cuando la relación identificada por el predicado sucede entre los objetos identificados por los términos. Las **sentencias compuestas** utilizan las conectivas como lo hace la lógica proposicional, y las **sentencias cuantificadas** permiten expresar reglas generales.
- Desarrollar una base de conocimiento en lógica de primer orden requiere un proceso cuidadoso para analizar el dominio, escoger el vocabulario y codificar los axiomas que se necesitan para soportar las inferencias deseadas.

Notas bibliográficas e históricas

Aunque incluso la lógica de Aristóteles trata con la generalización sobre los objetos, los inicios reales de la lógica de primer orden, con la introducción de los cuantificadores, se dan en el *Begriffschrift* («Escritura de Conceptos» o «Notación Conceptual») de Gottlob Frege (1879). La habilidad de Frege para anidar los cuantificadores fue un gran paso adelante, sin embargo, él utilizaba una notación algo poco elegante (un ejemplo aparece en la portada de este libro). La notación actual de la lógica de primer orden es sustancialmente de Giuseppe Peano (1889), pero la semántica es virtualmente idéntica a la de Frege. Aunque de manera extraña, los axiomas de Peano se debieron en gran medida la Grassmann (1861) y Dedekind (1888).

Una de las barreras más grandes en el desarrollo de la lógica de primer orden ha sido la concentración de esfuerzo en la lógica monádica respecto a la poliádica. Esta fijación sobre los predicados monádicos había sido casi universal en los sistemas lógicos desde Aristóteles hasta Boole. El primer tratamiento sistemático acerca de las relaciones lo realizó Augustus de Morgan (1864), quien citaba el siguiente ejemplo para mostrar los tipos de inferencia que la lógica de Aristóteles no podía manejar: «Todos los caballos son animales; por tanto, la cabeza de un caballo es la cabeza de un animal.» Esta inferencia es inaccesible a Aristóteles porque cualquier regla válida que puede apoyar esta inferencia primero debe analizar la sentencia utilizando el predicado binario «x es la cabeza de y». La lógica de las relaciones fue estudiada en profundidad por Charles Sanders Peirce (1870), quien también desarrolló la lógica de primer orden independientemente de Frege, aunque un poco más tarde (Peirce, 1883).

Leopold Löwenheim (1915) realizó un tratamiento sistemático de la teoría de modelos para la lógica de primer orden en 1915. Este texto trataba el símbolo de igualdad como una parte integral de la lógica. Los resultados de Löwenheim se desarrollaron bastante por Thoralf Skolem (1920). Alfred Tarski (1935, 1956) dio una definición explícita del valor de verdad y de la satisfacción en la teoría de modelos para la lógica de primer orden, utilizando la teoría de conjuntos.

McCarthy (1958) fue el principal responsable de la introducción de la lógica de primer orden como una herramienta para construir sistemas de IA. El porvenir de la IA basada en la lógica fue avanzando de forma significativa a partir del desarrollo de la resolución de Robinson (1965), un procedimiento completo de inferencia para la lógica de primer orden, que se describe en el Capítulo 9. El enfoque lógico asentó sus raíces en el Stanford. Cordell Green (1969a, 1969b) desarrolló un sistema de razonamiento en lógica de primer orden, QA3, llevando a los primeros intentos de construir un robot lógico en el SRI (Fikes y Nilsson, 1971). La lógica de primer orden se aplicó por Zohar Manna y Richard Waldinger (1971) para razonar acerca de los programas y más tarde por Michael Genesereth (1984) para razonar acerca de los circuitos. En Europa, la programación lógica (una forma restringida del razonamiento en lógica de primer orden) se desarrolló para el análisis lingüístico (Colmerauer *et al.*, 1973) y para sistemas declarativos en general (Kowalski, 1974). La lógica computacional también estuvo bien consolidada en Edinburgh a través del proyecto LCF (Lógica para Funciones Computables) de (Gordon *et al.*, 1979). En los Capítulos 9 y 10 se hace una crónica de estos desarrollos.

Hay un buen número de textos introductorios a la lógica de primer orden. Quine (1982) es uno de los más legibles. Enderton (1972) da una perspectiva orientada más a las matemáticas. Un tratamiento muy formal de la lógica de primer orden, a través de muchos y muy avanzados temas de la lógica, es el proporcionado por Bell y Machover (1977). Manna y Waldinger (1985) dan una introducción asequible a la lógica desde la perspectiva de las ciencias de la computación. Gallier (1986) proporciona una exposición matemática extremadamente rigurosa de la lógica de primer orden, a través del tratamiento de un enorme material sobre su uso en el razonamiento automático. *Logical Foundations of Artificial Intelligence* (Genesereth y Nilsson, 1987) proporciona tanto una introducción sólida a la lógica como el primer tratamiento sistemático de los agentes lógicos, con percepciones y acciones.

EJERCICIOS

8.1 Una base de conocimiento representa el mundo mediante un conjunto de sentencias con una estructura no explícita. Por otro lado, una representación **analógica** tiene una estructura física que se corresponde directamente con la estructura de lo que se representa. Considere un mapa de carreteras de su país como una representación analógica de hechos de ese país. La estructura bidimensional del mapa se corresponde con la estructura bidimensional de la superficie que se analiza.

a) Dé cinco ejemplos de *símbolos* para el lenguaje del mapa.

b) Una sentencia *explícita* es una sentencia que el creador de la representación realmente escribe. Una sentencia *implícita* es una sentencia que se obtiene de las sentencias explícitas a partir de las propiedades de la representación analógica. Dé tres ejemplos de sentencias *implícitas* y *explícitas* del lenguaje del mapa.

c) Dé tres ejemplos de hechos acerca de la estructura física de su país que no se pueden representar en el lenguaje del mapa.

d) Dé dos ejemplos de hechos que sean más fáciles de expresar en el lenguaje del mapa que en lógica de primer orden.

e) Dé dos ejemplos de representaciones analógicas útiles. ¿Cuáles son las ventajas y desventajas de cada uno de estos lenguajes?

8.2 Considere una base de conocimiento con tan sólo estas dos sentencias: $P(a)$ y $P(b)$. ¿Esta base de conocimiento implica $\forall x\, P(x)$? Explique su respuesta en términos de modelos.

8.3 ¿Es válida la sentencia $\exists x, y \;\; x = y$? Explíquelo.

8.4 Escriba una sentencia lógica tal que en cada mundo en que sea verdadera contenga exactamente un objeto.

8.5 Considere un vocabulario de símbolos que contenga símbolos de constante c, símbolos de predicado p_k para cada aridad k, y símbolos de función f_k para cada aridad k, donde $1 \leq k \leq A$. Permita que el tamaño del dominio esté fijado a D. Para una combinación dada de modelo-interpretación, cada símbolo de predicado o de función

es una aplicación a una relación o función, respectivamente, de la misma aridad. Puede asumir que las funciones en el modelo permiten algunas tuplas de entrada para las cuales la función no da ningún valor (por ejemplo, el valor es un objeto invisible). Derive una fórmula para el número de combinaciones modelo-interpretación para un dominio con D elementos. No se preocupe si debe eliminar combinaciones redundantes.

8.6 Represente las siguientes sentencias en lógica de primer orden, utilizando un vocabulario consistente (que usted debe definir):

a) Algunos estudiantes estudian francés en la primavera de 2001.
b) Cada estudiante que estudia francés lo aprueba.
c) Sólo un estudiante estudia griego en la primavera de 2001.
d) La mejor puntuación en griego siempre es mayor que la mejor puntuación en francés.
e) Todas las personas que compran una póliza son inteligentes.
f) Nadie compra una póliza cara.
g) Hay un agente que vende pólizas sólo a la gente que no está asegurada.
h) Hay un barbero que afeita a todos los hombres de la ciudad que no se afeitan ellos mismos.
i) Una persona nacida en Reino Unido, cuyos padres sean ciudadanos de Reino Unido o residentes en Reino Unido, es un ciudadano de Reino Unido.
j) Una persona nacida fuera de Reino Unido, que tenga uno de los padres ciudadano de Reino Unido o residente en Reino Unido, es ciudadano de Reino Unido por ascendencia.
k) Los políticos pueden mentir a algunos todo el tiempo, y pueden mentir a todos algún tiempo, pero no pueden mentir a todos todo el tiempo.

8.7 Represente la sentencia «Todos los alemanes hablan los mismos idiomas» en cálculo de predicados. Utilice *Habla*(x, l) para indicar que una persona x habla el idioma l.

8.8 ¿Qué axioma se necesita para inferir el hecho *Femenino*(*Laura*) dados los hechos *Masculino*(*Jim*) y *Esposo*(*Jim, Laura*)?

8.9 Escriba un conjunto general de hechos y axiomas para representar la aserción «Wellington oyó que Napoleón había muerto» y responda correctamente a la pregunta ¿Napoleón oyó que Wellington había muerto?

8.10 Transforme los hechos del mundo de *wumpus* en lógica proposicional de la Sección 7.5 a lógica de primer orden. ¿Cuánto más compacta es esta versión?

8.11 Escriba axiomas describiendo los predicados *Nieto, Bisabuelo, Hermano, Hermana, Hija, Hijo, Tía, Tío, HermanoPolítico, HermanaPolítica* y *PrimoHermano*. Averigüe la definición adecuada del primo n.º m, n veces extraído, y escriba la definición en lógica de primer orden.

Ahora anote los hechos básicos que están representados en el árbol familiar de la Figura 8.5. Mediante un sistema de razonamiento lógico apropiado, DILE todas las sentencias que ha anotado y PREGÚNTALE quién es el nieto de Elizabeth, los hermanos legales de Diana, y los bisabuelos de Zara.

Figura 8.5 Un árbol familiar típico. El símbolo «=» conecta esposos y las flechas apuntan a los hijos.

8.12 Anote una sentencia que aserte que la + es una función conmutativa. ¿Su sentencia se sigue de los axiomas de Peano? Si es así, explique por qué; y si no, dé un modelo en el que los axiomas sean verdaderos y su sentencia falsa.

8.13 Explique qué está incorrecto en la siguiente definición propuesta acerca del predicado de membresía a un conjunto, $\in$:

$$\forall x, s \;\; x \in \{x|s\}$$
$$\forall x, s \;\; x \in s \Rightarrow \forall y \;\; x \in \{y|s\}$$

8.14 Utilizando el conjunto de axiomas como ejemplo, escriba axiomas del dominio de las listas, incluyendo todas las constantes, funciones y predicados que se mencionan en el capítulo.

8.15 Explique qué está equivocado en la siguiente definición propuesta acerca de la adyacencia de casillas en el mundo de *wumpus*:

$$\forall x, y \;\; Adyacente([x, y], [x + 1, y]) \wedge Adyacente([x, y], [x, y + 1])$$

8.16 Escriba los axiomas que se necesitan para razonar acerca de la localización del *wumpus*, utilizando el símbolo de constante *wumpus* y un predicado binario *En*(*wumpus, Localización*). Recuerde que tan sólo hay un *wumpus*.

8.17 Amplíe el vocabulario de la Sección 8.4 para definir la adición de números binarios de n-bits. Entonces codifique la descripción del sumador de cuatro bits de la Figura 8.6, y plantee las peticiones que se necesitan para verificar que en efecto se comporta correctamente.

8.18 La representación del circuito en el capítulo es más detallado de lo que se necesita si sólo nos preocupa la funcionalidad del circuito. Una formulación más sencilla describe cualquier puerta, o circuito, de m-entradas y n-salidas, utilizando un predicado con $m + n$ argumentos, de tal manera que el predicado es verdadero cuando las entradas y las salidas son consistentes. Por ejemplo, las puertas NOT se describen mediante un predicado binario $NOT(i, o)$ para el cual $NOT(0, 1)$ y $NOT(1, 0)$ se conocen. Las composiciones de puertas se definen mediante conjunciones de predicados de puertas en las que

Figura 8.6 Un sumador de cuatro bits.

las variables compartidas indican conexiones directas. Por ejemplo, un circuito NAND puede estar compuesto por puertas *AND* y puertas *NOT*:

$$\forall i_1, i_2, o_a, o \;\; NAND(i_1, i_2, o) \Leftrightarrow AND(i_1, i_2, o_a) \wedge NOT(o_a, o)$$

Utilizando esta representación, defina el sumador de un bit de la Figura 8.4 y el sumador de cuatro bits de la Figura 8.6, y explique qué peticiones utilizaría para verificar los diseños. ¿Qué tipos de preguntas *no* se soportan mediante esta representación que *sí* se soportan mediante la representación de la Sección 8.4?

8.19 Obtenga una solicitud de pasaporte de su país, identifique las reglas que determinan la aprobación del pasaporte, y tradúzcalas a la lógica de primer orden, siguiendo los pasos perfilados en la Sección 8.4.

9 Inferencia en lógica de primer orden

Donde definiremos procedimientos eficientes para responder a preguntas planteadas en lógica de primer orden.

En el Capítulo 7 se definió el concepto de inferencia y se demostró cómo se puede llevar a cabo inferencia completa y sólida en lógica proposicional. En este capítulo ampliamos estos temas para conseguir algoritmos que pueden responder a preguntas expresadas en lógica de primer orden. Esto es muy significativo, porque si se trabaja duro, más o menos cualquier cosa se puede representar en lógica de primer orden.

En el Apartado 9.1 introducimos las reglas de inferencia para los cuantificadores y mostramos cómo reducir la inferencia de primer orden a la inferencia proposicional, aunque con un gran coste. En el Apartado 9.2 describimos el concepto de **unificación**, mostrando cómo se puede utilizar para construir reglas de inferencia que trabajen directamente en lógica de primer orden. Entonces discutimos sobre las tres grandes familias de algoritmos de inferencia de primer orden: en el Apartado 9.3 se tratan el **encadenamiento hacia delante** y sus aplicaciones en las **bases de datos deductivas** y en los **sistemas de producción**; en el Apartado 9.4 se desarrollan el **encadenamiento hacia atrás** y los sistemas de **programación lógica**; y en el Apartado 9.5 se describen los sistemas de **demostración de teoremas** basados en la resolución. Por lo general, uno intenta utilizar el método más eficiente que se pueda acomodar a los hechos y axiomas que se necesitan expresar. El razonamiento con sentencias totalmente generales en lógica de primer orden mediante la resolución suele ser menos eficiente que el razonamiento con cláusulas positivas mediante el encadenamiento hacia delante o hacia atrás.

9.1 Lógica proposicional vs. Lógica de primer orden

Esta sección y la siguiente introducen las ideas sobre las que se basan los sistemas actuales de inferencia lógicos. Comenzamos con algunas reglas de inferencia sencillas que se pueden aplicar a las sentencias con cuantificadores para obtener sus sentencias equivalentes, sin cuantificar. Estas reglas nos conducen de forma natural a la idea de que la inferencia *de primer orden* se puede hacer convirtiendo la base de conocimiento a lógica *proposicional* y utilizando la inferencia *proposicional*. La siguiente sección nos muestra un atajo que es obvio, ir hacia los métodos de inferencia que manipulen directamente las sentencias en lógica de primer orden.

Reglas de inferencia para cuantificadores

Vamos a comenzar con los cuantificadores universales. Suponga que nuestra base de conocimiento contiene el axioma popular que afirma que los reyes que son codiciosos también son malvados.

$$\forall x \;\; Rey(x) \wedge Codicioso(x) \Rightarrow Malvado(x).$$

Entonces parece bastante permisible inferir cualquiera de las siguientes sentencias:

$$\begin{array}{l} Rey(Juan) \wedge Codicioso(Juan) \Rightarrow Malvado(Juan). \\ Rey(Ricardo) \wedge Codicioso(Ricardo) \Rightarrow Malvado(Ricardo). \\ Rey(Padre(Juan)) \wedge Codicioso(Padre(Juan)) \Rightarrow Malvado(Padre(Juan)). \\ \quad \vdots \end{array}$$

ESPECIFICACIÓN UNIVERSAL

La regla de la **Especificación Universal** (EU para abreviar) dice que podemos inferir cualquier sentencia obtenida por sustitución de la variable por un **término base** (un término sin variables)[1]. Para anotar la regla de inferencia formalmente utilizamos el concepto de **sustitución** que se introdujo en el Apartado 8.3. Vamos a denotar el resultado de aplicar una sustitución θ a una sentencia α mediante $\textsc{Sust}(\theta, \alpha)$. Entonces la regla se escribe

$$\frac{\forall v \;\; \alpha}{\textsc{Sust}(\{v/g\}, \alpha)}$$

para cualquier variable v y término base g. Por ejemplo, las tres sentencias mostradas anteriormente se obtienen con las sustituciones $\{x/Juan\}$, $\{x/Ricardo\}$, y $\{x/Padre(Juan)\}$.

ESPECIFICACIÓN EXISTENCIAL

La correspondiente regla de **Especificación Existencial** para el cuantificador existencial es ligeramente más complicada. Para cualquier sentencia α, variable v, y símbolo de constante k que no aparezca en ninguna otra parte de la base de conocimiento,

$$\frac{\exists v \;\; \alpha}{\textsc{Sust}(\{v/k\}, \alpha)}$$

[1] No confunda estas sustituciones con las interpretaciones ampliadas utilizadas para definir la semántica de los cuantificadores. La sustitución reemplaza una variable por un término (una pieza de la sintaxis) para producir una nueva sentencia, mientras que una interpretación es una aplicación de una variable a un objeto del dominio de discurso.

Por ejemplo, la sentencia

$$\exists x \;\; Corona(x) \wedge SobreCabeza(x, Juan)$$

podemos inferir la sentencia

$$Corona(C_1) \wedge SobreCabeza(C_1, Juan)$$

mientras que C_1 no aparezca en ningún otro sitio de la base de conocimiento. Básicamente, la sentencia existencial nos dice que hay algún objeto que satisface una condición, y el proceso de especificación tan sólo le da un nombre a dicho objeto. Naturalmente, ese nombre no puede pertenecer a otro objeto previamente. Las matemáticas nos proporcionan un delicioso ejemplo: suponga que descubrimos que hay un número que es un poco mayor que 2,71828 y que satisface la ecuación $d(x^y)/dy = x^y$ para x. Le podemos dar a dicho número un nombre, como e, pero sería un error darle el nombre de un objeto ya existente, como π. En lógica, a este nuevo nombre se le denomina **constante de Skolem**. La Especificación Existencial es un caso especial del proceso más general llamado **skolemización**, que trataremos en el Apartado 9.5.

CONSTANTE DE SKOLEM

Así como es más complicada que la Especificación Universal, la Especificación Existencial representa un papel algo diferente en la inferencia. Mientras que la Especificación Universal se puede aplicar muchas veces para producir muchas consecuencias diferentes, la Especificación Existencial sólo se puede aplicar una vez, y entonces se puede descartar la sentencia cuantificada existencialmente. Por ejemplo, una vez que hemos añadido la sentencia $Mata(Asesino, Víctima)$, ya no necesitamos más la sentencia $\exists x\; Mata(x, Víctima)$. Hablando de forma estricta, la nueva base de conocimiento no es equivalente lógicamente a la antigua, pero se puede demostrar que es **equivalente inferencialmente** en el sentido que es *satisfacible* justamente cuando lo es la base de conocimiento original.

EQUIVALENCIA INFERENCIAL

Reducción a la inferencia proposicional

Una vez que tenemos las reglas para inferir sentencias no cuantificadas a partir de sentencias cuantificadas, nos es posible reducir la inferencia de primer orden a la inferencia proposicional. En esta sección daremos las principales ideas; los detalles se verán en el Apartado 9.5.

La primera idea consiste en que como una sentencia cuantificada existencialmente se puede sustituir por una especificación, una sentencia cuantificada existencialmente se puede sustituir por el conjunto de *todas las especificaciones posibles*. Por ejemplo, suponga que nuestra base de conocimiento contiene tan sólo las sentencias

$$\begin{aligned} &\forall x \;\; Rey(x) \wedge Codicioso(x) \Rightarrow Malvado(x) \\ &Rey(Juan) \\ &Codicioso(Juan) \\ &Hermano(Ricardo, Juan) \end{aligned} \tag{9.1}$$

Entonces aplicamos la EU a la primera sentencia, utilizando todas las sustituciones de términos base posibles, tomadas del vocabulario de la base de conocimiento, en este caso {x/Juan} y {x/Ricardo}. Obtenemos

$$\begin{aligned} &Rey(Juan) \wedge Codicioso(Juan) \Rightarrow Malvado(Juan), \\ &Rey(Ricardo) \wedge Codicioso(Ricardo) \Rightarrow Malvado(Ricardo), \end{aligned}$$

y descartamos la sentencia cuantificada universalmente. Ahora, la base de conocimiento es esencialmente proposicional, si vemos las sentencias atómicas base (*Rey*(*Juan*), *Codicioso*(*Juan*), etc.) como símbolos proposicionales. Por tanto, podemos aplicar cualquiera de los algoritmos proposicionales completos del Capítulo 7, para obtener conclusiones como *Malvado*(*Juan*).

PROPOSICIONALIZACIÓN

Esta técnica de **proposicionalización** se puede hacer que sea completamente general, tal como mostraremos en el Apartado 9.5; es decir, toda base de conocimiento, y petición, en lógica de primer orden se puede transformar a forma proposicional de tal manera que se mantenga la relación de implicación. De este modo, tenemos un procedimiento de decisión completo para la implicación… o quizá no. Hay un problema: ¡cuando la base de conocimiento incluye un símbolo de función, el conjunto de sustituciones de los términos base es infinito! Por ejemplo, si la base de conocimiento tiene el símbolo *Padre*, entonces se pueden construir términos anidados infinitamente, como *Padre*(*Padre*(*Padre*(*Juan*))). Nuestros algoritmos proposicionales tendrían serias dificultades con tal conjunto de sentencias infinitamente grande.

Afortunadamente, hay un famoso teorema de Jacques Herbrand (1930) que dice que si una sentencia se implica de una base de conocimiento de primer orden, entonces hay una demostración que involucra tan sólo a un subconjunto finito de la base de conocimiento transformada a proposicional. Ya que cada subconjunto de este tipo tiene un máximo de profundidad en la anidación de sus términos base, podemos encontrar el subconjunto generando primero todas las especificaciones al nivel de los símbolos de constante (*Ricardo* y *Juan*), luego con el nivel siguiente de profundidad, o profundidad 1, (*Padre*(*Ricardo*) y *Padre*(*Juan*), luego con los de nivel de profundidad 2, etc., hasta que seamos capaces de construir una demostración proposicional de la sentencia implicada.

Hemos esbozado un enfoque de inferencia en lógica de primer orden mediante la proposicionalización que es **completo**, es decir, cualquier sentencia implicada se puede demostrar. Esto es un importante logro, dado que el espacio de los modelos posibles es infinito. Por otro lado, ¡no podemos saber que la sentencia *se* implica hasta que la demostración se ha realizado! ¿Qué ocurre cuando la sentencia no se implica? ¿Podemos decir algo? Bueno, en lógica de primer orden, resulta que no podemos. Nuestro procedimiento de demostración podría continuar, generando más y más términos profundamente anidados, pero no sabremos si se ha atascado en un bucle inútil o si la demostración está a punto de acabar. Algo que se parece mucho al problema de la parada en las máquinas de Turing. Alan Turing (1936) y Alonzo Church (1936) demostraron, mediante caminos más bien diferentes, lo inevitable de este tipo de cosas. *El problema de la implicación en lógica de primer orden es* ***semidecidible****, es decir, existen algoritmos que responden afirmativamente para cada sentencia implicada, pero no existe ningún algoritmo que también responda ante una sentencia no implicada.*

9.2 Unificación y sustitución

La sección anterior describía cómo la comprensión de la inferencia en lógica de primer orden parte de los inicios de los 60. El lector observador (y seguramente los lógicos com-

putacionales de los inicios de los 60) se habrá dado cuenta de que el enfoque de la proposicionalización es más bien ineficiente. Por ejemplo, dada la petición *Malvado*(x) y la base de conocimiento de la Ecuación (9.1) parece algo obstinado generar sentencias como $Rey(Juan) \wedge Codicioso(Juan) \Rightarrow Malvado(Juan)$. Sin embargo, la inferencia de *Malvado*(*Juan*) a partir de las sentencias

$$\begin{array}{l} \forall x \;\; Rey(x) \wedge Codicioso(x) \Rightarrow Malvado(x) \\ Rey(Juan) \\ Codicioso(Juan) \end{array}$$

parece completamente obvio para un ser humano. Ahora mostraremos cómo hacerlo completamente obvio para un computador.

Una regla de inferencia de primer orden

La inferencia de que Juan es malvado se obtiene de la siguiente manera: encontrar un x tal que x sea rey y x sea codicioso, y entonces inferir que ese x es malvado. De forma más general, si hay alguna sustitución θ que haga que las premisas de la implicación sean idénticas a algunas de las sentencias que ya están en la base de conocimiento, entonces podemos asertar la conclusión de la implicación, después de aplicar θ. En este caso, la sustitución $\{x/Juan\}$ logra ese objetivo.

En realidad podemos hacer que el paso de inferencia aún trabaje más. Suponga que en vez de conocer *Codicioso*(*Juan*), sabemos que *todo el mundo* es codicioso:

$$\forall y \;\; Codicioso(y) \tag{9.2}$$

entonces también seríamos capaces de concluir que *Malvado*(*Juan*), porque sabemos que Juan es un rey (dado) y que Juan es codicioso (porque todo el mundo lo es). Lo que necesitamos para este trabajo es encontrar una sustitución para las variables en la sentencia de implicación y las variables de las sentencias que se deben emparejar. En este caso, aplicando la sustitución $\{x/Juan, y/Juan\}$ a las premisas de la implicación $Rey(x)$ y $Codicioso(x)$ y a las sentencias de la base de conocimiento $Rey(Juan)$ y $Codicioso(y)$ las hará idénticas. De este modo podemos inferir la conclusión de la implicación.

MODUS PONENS GENERALIZADO

Este proceso de inferencia se puede plasmar mediante una única regla de inferencia a la que llamamos **Modus Ponens Generalizado**: para las sentencias atómicas p_i, p_i', y q, donde hay una sustitución θ tal que $\text{SUST}(\theta, p_i') = \text{SUST}(\theta, p_i)$, para todo i,

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUST}(\theta, q)}$$

Hay $n + 1$ premisas para esta regla: las n p_i' sentencias atómicas y la implicación. La conclusión es el resultado de aplicar la sustitución θ al consecuente q. En nuestro ejemplo:

$$\begin{array}{ll} p_1' \text{ es } Rey(Juan) & p_1 \text{ es } Rey(x) \\ p_2' \text{ es } Codicioso(y) & p_2 \text{ es } Codicioso(x) \\ \theta \text{ es } \{x/Juan, y/Juan\} & q \text{ es } Malvado(x) \\ \text{SUST}(\theta, q) \text{ es } Malvado(Juan) & \end{array}$$

Es fácil demostrar que el Modus Ponens Generalizado es una regla de inferencia sólida. Primero, observamos que cualquier sentencia p (cuyas variables asumimos que están cuantificadas universalmente) y para cualquier sustitución θ,

$$p \models \text{SUST}(\theta, p)$$

Esto se sostiene sobre los mismos fundamentos de la Especificación Universal. Y se sostiene en concreto para una sustitución θ que satisfaga las condiciones de la regla del Modus Ponens Generalizado. De este modo, de $p_1', \ldots, p_n'$ podemos inferir

$$\text{SUST}(\theta, p_1') \wedge \ldots \wedge \text{SUST}(\theta, p_n')$$

y de la implicación $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$ podemos inferir

$$\text{SUST}(\theta, p_1) \wedge \ldots \wedge \text{SUST}(\theta, p_n) \Rightarrow \text{SUST}(\theta, q)$$

Ahora bien, θ se define en el Modus Ponenes Generalizado como $\text{SUST}(\theta, p_i') = \text{SUST}(\theta, p_i)$, para todo i; por lo tanto la primera sentencia se empareja exactamente con la premisa de la segunda. De aquí, que $\text{SUST}(\theta, q)$ se siga del Modus Ponens.

ELEVACIÓN

El Modus Ponens Generalizado es una versión **elevada** del Modus Ponens: erige el Modus Ponens proposicional a la lógica de primer orden. En lo que queda del capítulo veremos que podemos desarrollar versiones elevadas del encadenamiento hacia delante, del encadenamiento hacia atrás, y de los algoritmos de resolución que se introdujeron en el Capítulo 7. La ventaja clave de las reglas de inferencia elevadas sobre la proposicionalización es que sólo realizan aquellas sustituciones que se necesitan para permitir avanzar a las inferencias. Un tema potencialmente confuso es que uno percibe que el Modus Ponens Generalizado es menos general que el Modus Ponens (Apartado 7.5): el Modus Ponens reconoce cualquier α atómica que esté en el lado izquierdo de la implicación, mientras que el Modus Ponens Generalizado necesita un formato especial para esta sentencia. La regla es generalizada en el sentido de que permite trabajar con cualquier número de P_i'.

Unificación

UNIFICACIÓN

UNIFICADOR

Las reglas de inferencia elevadas necesitan encontrar las sustituciones que hacen que expresiones lógicas diferentes se hagan idénticas. Este proceso se denomina **unificación** y es el componente clave de todos los algoritmos de inferencia en lógica de primer orden. El algoritmo UNIFICA toma dos sentencias y devuelve un **unificador** para ellas, si éste existe:

$$\text{UNIFICA}(p, q) = \theta \text{ donde } \text{SUST}(\theta, p) = \text{SUST}(\theta, q).$$

Vamos a ver algunos ejemplos de cómo se comportaría UNIFICA. Suponga que tenemos una petición *Conoce*(*Juan*, *x*): ¿a quién conoce Juan? Algunas respuestas a esta pregunta se pueden hallar encontrando todas las sentencias de la base de conocimiento que se unifiquen con *Conoce*(*Juan*, *x*). Aquí tenemos los resultados de la unificación con cuatro sentencias distintas que podrían estar en la base de conocimiento.

UNIFICA(*Conoce*(*Juan*, *x*), *Conoce*(*Juan*, *Juana*)) = {*x*/*Juana*}
UNIFICA(*Conoce*(*Juan*, *x*), *Conoce*(*y*, *Guillermo*)) = {*Guillermo, y*/*Juan*}
UNIFICA(*Conoce*(*Juan*, *x*), *Conoce*(*y*, *Madre*(*y*))) = {*y*/*Juan*, *x*/*Madre*(*Juan*)}
UNIFICA(*Conoce*(*Juan*, *x*), *Conoce*(*x*, *Elisabet*)) = *fallo*.

La última unificación falla porque *x* no puede tomar los valores *Juan* y *Elisabet* al mismo tiempo. Ahora, recuerda que *Conoce*(*x*, *Elisabet*) significa «Todo el mundo conoce a Elisabet», por tanto, seríamos capaces de inferir que Juan conoce a Elisabet. El problema se presenta sólo cuando las dos sentencias tienen que utilizar el mismo nombre de variable, *x*. Este problema se puede evitar utilizando la **estandarización de las variables** de las sentencias que van a ser unificadas, que consiste en renombrar sus variables para evitar conflictos de nombre. Por ejemplo, podemos renombrar la *x* de *Conoce*(*x*, *Elisabet*) por z_{17} (un nuevo nombre de variable) sin tener que cambiar su significado. Ahora la unificación tendrá éxito:

ESTANDARIZACIÓN DE VARIABLES

UNIFICA(*Conoce*(*Juan*, *x*), *Conoce*(z_{17}, *Elisabet*)) = {*x*/*Elisabet*, z_{17}/*Juan*}

El Ejercicio 9.7 ahonda más en la necesidad de la estandarización de variables.

Hay una complicación más: decimos que UNIFICA debería devolver una sustitución que hace que dos argumentos sean idénticos. Pero podría haber más de un unificador. Por ejemplo, UNIFICA(*Conoce*(*Juan*, *x*), *Conoce*(*y*, *z*)) podría devolver {*y*/*Juan*, *x*/*z*} o {*y*/*Juan*, *x*/*Juan*, *z*/*Juan*}. El primer unificador nos da *Conoce*(*Juan*, *z*) como resultado de la unificación, mientras que el segundo nos da *Conoce*(*Juan*, *Juan*); decimos que el primer unificador es más *general* que el segundo, porque coloca menos restricciones sobre los valores de las variables. Resulta que, para cada par de expresiones unificable hay un **unificador más general** (o UMG) que es único respecto al renombramiento de las variables. En este caso {*y*/*Juan*, *x*/*z*}.

UNIFICADOR MÁS GENERAL

En la Figura 9.1 se muestra un algoritmo para obtener el unificador más general. El proceso es muy sencillo: el algoritmo explora recursivamente las dos expresiones «de lado a lado», acumulando un unificador durante el proceso, pero falla si dos posiciones de las dos estructuras que se corresponden no emparejan. Hay un paso en el proceso que es costoso: cuando al intentar emparejar una variable con un término complejo, se debe comprobar si la propia variable ya pertenece al término; y si es así, el emparejamiento falla porque no se puede generar un unificador consistente. Este proceso, denominado **comprobación de ocurrencias**, hace que la complejidad del algoritmo sea cuadrática respecto al tamaño de las expresiones que se van a unificar. Algunos sistemas, incluidos todos los sistemas de programación lógica, simplemente omiten este proceso por lo que algunas veces obtienen como resultado inferencias inconsistentes; otros sistemas utilizan algoritmos más complejos que presentan una complejidad temporal de tipo lineal.

COMPROBACIÓN DE OCURRENCIAS

Almacenamiento y recuperación

Por debajo de las funciones DECIR y PREGUNTAR, utilizadas para informar o interrogar a la base de conocimiento, están las funciones primitivas ALMACENAR y BUSCAR. ALMACENAR (*s*) guarda una sentencia *s* en la base de conocimiento, y BUSCAR(*q*) devuelve todos los unificadores que unifican la petición *q* con alguna sentencia de la base

```
función UNIFICA(x, y, θ) devuelve una sustitución que hace x e y idénticas
   entradas: x, una variable, constante, lista, o expresión compuesta
             y, una variable, constante, lista, o expresión compuesta
             θ, la sustitución construida hasta ahora (opcional, por defecto está vacía)

   si θ = fallo entonces devolver fallo
   sino si x = y entonces devolver θ
   sino si ¿VARIABLE?(x) entonces devolver UNIFICA-VAR(x, y, θ)
   sino si ¿VARIABLE?(y) entonces devolver UNIFICA-VAR(y, x, θ)
   sino si ¿EXP-COMPUESTA?(x) y ¿EXP-COMPUESTA?(y) entonces
      devolver UNIFICA(ARGS[x], ARGS[y], UNIFICA(OP[x], OP[y], θ))
   sino si ¿LISTA?(x) y ¿LISTA?(y) entonces
      devolver UNIFICA(REST[x], REST[y], UNIFICA(PRIMERO[x], PRIMERO[y], θ))
   sino devolver fallo
```

```
Función UNIFICA-VAR(var, x, θ) devuelve una sustitución
   Entradas: var, una variable
             x, una expresión
             θ, la sustitución construida hasta ahora

   si {var/val} ∈ θ entonces devolver UNIFICA(val, x, θ)
   sino si {x/val} ∈ θ entonces devolver UNIFICA(var, val, θ)
   sino si ¿COMPRUEBA-OC?(var, x) entonces devolver fallo
   sino devolver añadir {var/x} a θ
```

Figura 9.1 El algoritmo de unificación. El algoritmo trabaja mediante la comparación, elemento a elemento, de las expresiones de entrada. La sustitución θ, que es el argumento de UNIFICA, se va construyendo a lo largo de todo el proceso y se utiliza para asegurar que las comparaciones posteriores sean consistentes con las ligaduras que previamente se han establecido. En una expresión compuesta, como $F(A, B)$, la función OP selecciona el símbolo de función F, y la función ARGS selecciona la lista de argumentos (A, B).

de conocimiento. El problema que hemos utilizado para ilustrar la unificación (encontrando todos los hechos que se unifican con *Conoce*(*Juan*, *x*)) es una instancia de BUSCAndo.

La forma más sencilla de implementar ALMACENAR y BUSCAR consiste en mantener todos los hechos en la base de conocimiento mediante una lista larga; entonces, dada una petición *q*, se llama a UNIFICA(*q*, *s*) con cada sentencia s de la lista. Este proceso es ineficiente, pero trabaja bien, y es todo lo que necesita entender para el resto del capítulo. Lo que queda de esta sección da una idea general de los mecanismos para realizar la recuperación de forma más eficiente, y por lo tanto, en una primera lectura se lo puede saltar.

INDEXANDO

INDEXACIÓN DE PREDICADOS

Podemos BUSCAR de forma más eficiente asegurándonos de que las unificaciones sólo se intenten con las sentencias que tienen *alguna* oportunidad de unificarse. Por ejemplo, no hay ninguna posición que pueda unificar *Conoce*(*Juan*, *x*) y *Hermano*(*Ricardo*, *Juan*). Podemos evitar este tipo de unificaciones **indexando** los hechos en la base de conocimiento. Un sencillo programa, denominado **indexación de predicados**, coloca to-

dos los hechos *Conoce* en un cajón y todos los hechos *Hermano* en otro. Los cajones se pueden almacenar en una tabla *hash*[2] para obtener un acceso más eficiente.

La indexación de predicados es útil cuando hay muchos símbolos de predicado pero sólo unas pocas cláusulas por cada símbolo. En algunas aplicaciones, hay muchas cláusulas para un símbolo de predicado dado. Por ejemplo, suponga que las entidades de recaudación de impuestos quieren guardar la pista de quién contrata a quién, utilizando el predicado *Contrata*(x, y). Esto generaría un cajón muy grande con quizá millones de contratantes y decenas de millones de contratados. Y por tanto, responder a una petición como *Contrata*(x, *Ricardo*) mediante la indexación de predicados requeriría recorrer el cajón entero.

Para esta petición en concreto, sería de ayuda que los hechos estuvieran indexados tanto por el predicado como por su segundo argumento, quizás utilizando una tabla *hash* de claves combinadas. Entonces podríamos simplemente construir la clave para la petición y recuperar exactamente aquellos hechos que se unifican con la petición. Para otras peticiones, tales como *Contrata*(*AIMA.org*, y), necesitaríamos haber indexado los hechos combinando el predicado con el primer argumento. Por lo tanto, los hechos se pueden almacenar bajo múltiples claves, haciéndolos accesibles instantáneamente a las diferentes peticiones con las que se podrían unificar.

Dada una sentencia a almacenar, es posible construir los índices para *todas las posibles* peticiones que se unifican con ella. Para el hecho *Contrara*(*AIMA.org*, *Ricardo*), las peticiones son

Contrata(*AIMA.org*, *Ricardo*)	¿Contrata AIMA a Ricardo?
Contrata(x, *Ricardo*)	¿Quién contrata a Ricardo?
Contrata(*AIMA.org*, y)	¿A quién contrata AIMA.org?
Contrata(x, y)	¿Quién contrata a quién?

RETÍCULO DE SUBSUNCIÓN

Estas peticiones forman un **retículo de subsunción**, tal como se muestra en la Figura 9.2(a). El retículo tiene algunas propiedades interesantes. Por ejemplo, el hijo de cualquier nodo del retículo se obtiene de su padre mediante una subsunción única; y el descendiente común «más alto» de dos nodos cualesquiera es el resultado de aplicar su unificador más general. La porción del retículo por encima de cualquier hecho base se puede construir sistemáticamente (Ejercicio 9.5). Una sentencia con constantes repetidas tiene un retículo algo diferente, tal como se muestra en la Figura 9.2(b). Los símbolos de función y las variables en las sentencias a que pueden ser almacenadas aún generan estructuras de retículos más interesantes.

El programa que hemos descrito trabaja muy bien siempre que el retículo contenga un número pequeño de nodos. Para un predicado con n argumentos, el retículo contiene $O(2^n)$ nodos. Si se permiten los símbolos de función el número de nodos entonces es exponencial respecto al número de términos de la sentencia a almacenar. Esto nos puede conducir a un número inmenso de índices. En algún punto, los beneficios de la indexación son superados por los costes de almacenar y mantener todos los índices. Podemos responder adoptando una política fija, tal como mantener sólo los índices de las claves compuestas por un predicado

[2] Una tabla *hash* es una estructura de datos para almacenar y recuperar información indexada mediante *claves* fijas. Por motivos prácticos, se puede considerar que una tabla *hash* tiene tiempos constantes de almacenamiento y recuperación, aun cuando la tabla contenga un número enorme de elementos.

Figura 9.2 (a) El retículo de subsunción cuyo nodo más bajo es la sentencia *Contrata*(*AIMA.org, Ricardo*). (b) El retículo de subsunción para la sentencia *Emplea*(*Juan, Juan*).

y cada uno de sus argumentos, o utilizando una política adaptativa que genere índices que respondan a las demandas de los tipos de peticiones que se desean realizar. En muchos sistemas de IA, el número de los hechos que se almacenan es lo suficientemente pequeño de manera que la indexación eficiente se considera un problema resuelto. En las bases de datos industriales y comerciales, el problema ha recibido un desarrollo tecnológico sustancial.

9.3 Encadenamiento hacia delante

En el Apartado 7.5 se dio un algoritmo de encadenamiento hacia delante para cláusulas positivas proposicionales. La idea es simple: comenzar a partir de las sentencias atómicas de la base de conocimiento y aplicar el Modus Ponens hacia delante, añadiendo las sentencias atómicas nuevas hasta que no se puedan realizar más inferencias. Aquí explicamos cómo se puede aplicar el algoritmo a las cláusulas positivas de primer orden, y cómo se puede implementar eficientemente. Las cláusulas positivas como *Situación* $\Rightarrow$ *Respuesta* son especialmente útiles para los sistemas que realizan inferencias en respuesta a información nueva que se ha recibido. Muchos sistemas se pueden definir de esta manera, y el razonamiento hacia delante puede ser mucho más eficiente que la resolución aplicada a la demostración de problemas. Por lo tanto, a menudo vale la pena intentar construir una base de conocimiento tan sólo con cláusulas positivas de tal manera que se evite el coste de aplicar la resolución.

Cláusulas positivas de primer orden

Las cláusulas positivas de primer orden se parecen bastante a las cláusulas positivas proposicionales (Apartado 7.5): éstas son disyunciones de literales de los cuales *sólo uno es positivo*. Una cláusula positiva es atómica o es una implicación cuyo antecedente es una conjunción de literales positivos y cuyo consecuente es un único literal positivo. Las siguientes son clásusulas positivas de primer orden:

$$Rey(x) \wedge Codicioso(x) \Rightarrow Malvado(x)$$
$$Rey(Juan)$$
$$Codicioso(y)$$

Diferentes a los literales proposicionales, los literales de primer orden pueden contener variables, en cuyo caso se asume que dichas variables están cuantificadas universalmente. (Lo típico es que omitamos los cuantificadores universales cuando escribamos cláusulas positivas.) Las cláusulas positivas son una forma normal muy adecuada para utilizarla con el Modus Ponens Generalizado.

No se pueden convertir todas las bases de conocimiento a un conjunto de cláusulas positivas, por la restricción de un único literal positivo, pero muchas sí se pueden transformar. Considere el siguiente problema:

> La ley dice que es un crimen para un americano vender armas a países hostiles. El país de Nono, un enemigo de América, tiene algunos misiles, y todos sus misiles fueron vendidos por el Coronel West, que es americano.

Demostraremos que West es un criminal. Primero, representaremos estos hechos mediante cláusulas positivas de primer orden. La siguiente sección mostrará cómo el encadenamiento hacia delante resuelve este problema.

«… es un crimen para un americano vender armas a países hostiles»:

$$Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminal(x) \qquad (9.3)$$

«Nono …tiene algunos misiles.» La sentencia $\exists x\ Tiene(Nono, x) \wedge Misil(x)$ se transforma en dos cláusulas positivas mediante la Eliminación del Existencial, introduciendo la nueva constante M_1:

$$Tiene(Nono, M_1) \qquad (9.4)$$
$$Misil(M_1) \qquad (9.5)$$

«Todos los misiles le (a Nono) fueron vendidos por el coronel West»:

$$Misil(x) \wedge Tiene(Nono, x) \Rightarrow Vende(West, x, Nono) \qquad (9.6)$$

También necesitamos saber que los misiles son armas:

$$Misil(x) \Rightarrow Arma(x) \qquad (9.7)$$

Y necesitamos saber que un enemigo de América es un país «hostil»:

$$Enemigo(x, América) \Rightarrow Hostil(x) \qquad (9.8)$$

«West, que es Americano…»:

$$Americano(West) \qquad (9.9)$$

«El país Nono, un enemigo de América…»:

$$Enemigo(Nono, América) \qquad (9.10)$$

DATALOG

Esta base de conocimiento no contiene símbolos de función y es por lo tanto una instancia de la clase de bases de conocimiento de **Datalog**, es decir, conjuntos de cláusulas positivas de primer orden sin símbolos de función. Veremos que la ausencia de los símbolos de función hace mucho más fácil la inferencia.

Un algoritmo sencillo de encadenamiento hacia delante

El primer algoritmo de encadenamiento hacia delante que consideraremos es uno muy sencillo, tal como se muestra en la Figura 9.3. Comenzando con los hechos conocidos, el proceso dispara todas las reglas cuyas premisas se satisfacen, añadiendo sus conclusiones al conjunto de hechos conocidos. El proceso se va repitiendo hasta que la petición es respondida (asumiendo que sólo se requiere una respuesta) o no se pueden añadir más hechos. Fíjese en que un hecho no es «nuevo», sólo es el **renombramiento** de un hecho conocido. Una sentencia es el renombramiento de otra si son idénticas excepto en los nombres de las variables. Por ejemplo, *Gusta*(*x*, *Helado*) y *Gusta*(*y*, *Helado*) son renombramientos cada una de la otra porque sólo se diferencian en la elección de *x* o de *y*; sus significados son el mismo: a todo el mundo le gusta el helado.

RENOMBRAR VARIABLES

Utilizaremos nuestro problema sobre el crimen para ilustrar cómo funciona PREGUNTA-EHD-LPO. Las sentencias de implicación son (9.3), (9.6), (9.7) y (9.8). Se necesitan dos iteraciones:

- En la primera iteración, la regla (9.3) no tiene las premisas satisfechas.
  La regla (9.6) se satisface con $\{x/M_1\}$, y se añade *Vende*(*West*, M_1, *Nono*).
  La regla (9.7) se satisface con $\{x/M_1\}$, y se añade *Arma*(M_1).
  La regla (9.8) se satisface con $\{x/Nono\}$, y se añade *Hostil*(*Nono*).
- En la segunda iteración, la regla (9.3) se satisface con $\{x/West, y/M_1, z/Nono\}$, y se añade *Criminal*(*West*).

```
función PREGUNTA-EHD-LPO(BC, α) devuelve una sustitución o falso
  entradas: BC, la base de conocimiento, un conjunto de cláusulas positivas de primer orden
            α, la petición, una sentencia atómica
  variables locales: nuevas, las nuevas sentencias inferidas en cada iteración

repetir hasta nuevo está vacío
    nuevo ← { }
    para cada sentencia r en BC hacer
      (p1 ∧ … ∧ pn ⇒ q) ← ESTANDARIZAR-VAR(r)
      para cada θ tal que SUST(θ, p1 ∧ … ∧ pn) = SUST(p'1 ∧ … ∧ p'n)
                  para algún p'1 … p'n en BC
        q' ← SUST(θ, q)
        si q' no es el renombramiento de una sentencia de BC o nuevo entonces hacer
        añadir q' a nuevo
        φ ← UNIFICA(q', α)
        si φ no es fallo entonces devolver φ
    añadir nuevo a BC
  devolver falso
```

Figura 9.3 Un algoritmo de encadenamiento hacia delante, conceptualmente sencillo, pero muy ineficiente. En cada iteración añade a la *BC* todas las sentencias atómicas que se pueden inferir en un paso, a partir de las sentencias de implicación y las sentencias atómicas ya presentes en la *BC*.

Figura 9.4 El árbol de demostración generado por el encadenamiento hacia delante en el ejemplo del crimen. Los hechos iniciales aparecen en el nivel inferior, los hechos inferidos en la primera iteración aparecen en el nivel intermedio, y los hechos inferidos en la segunda iteración aparecen en el nivel superior.

La Figura 9.4 muestra el árbol de demostración que se ha generado. Fíjese en que las nuevas inferencias son posibles en este punto porque cada sentencia que podía concluirse mediante el encadenamiento hacia delante ya está contenida explícitamente en la *BC*. Este tipo de base de conocimiento se dice que tiene un proceso de inferencia de **punto fijo**. Los puntos fijos hallados con las cláusulas positivas de primer orden son similares a aquellas que se hallan en el encadenamiento hacia delante proposicional (Apartado 7.5); la principal diferencia es que el punto fijo de primer orden puede incluir sentencias atómicas cuantificadas universalmente.

Es fácil analizar el algoritmo PREGUNTA-EHD-LPO. Primero, es un algoritmo **sólido**, porque cada inferencia es justo una aplicación del Modus Ponens Generalizado, que es sólida. Segundo, es **completo** para las bases de conocimiento con cláusulas positivas; es decir, responde cada petición cuyas respuestas se implican de cada base de conocimiento con cláusulas positivas. En las bases de conocimiento Datalog, que no contienen símbolos de función, la demostración de su completitud es bastante fácil. Comenzamos contando el número de los posibles hechos que pueden ser añadidos, lo que determina el número máximo de iteraciones. Siendo k la **aridad** máxima (número de argumentos) de cada predicado, p el número de predicados, y n el número de símbolos de constante. Está claro que no puede haber más de pn^k hechos base, así que después de estas muchas iteraciones el algoritmo debe encontrar un punto fijo. Entonces podemos construir un argumento muy similar a la demostración de la completitud en el encadenamiento hacia delante proposicional (Apartado 7.5). Los detalles acerca de la transición de la completitud proposicional a la de primer orden se dan en el algoritmo de resolución del Apartado 9.5.

Con las cláusulas positivas que contienen símbolos de función, PREGUNTA-EHD-LPO puede generar muchos hechos nuevos infinitamente, por lo que necesitamos tener mucho cuidado. Para el caso en el que una respuesta se implica para la sentencia de petición q, debemos recurrir al teorema de Herbrand para establecer que el algoritmo

encontrará una demostración. (*Véase* el Apartado 9.5 para el caso de la resolución.) Si la petición no tiene respuesta, el algoritmo podría fallar y terminar en algunos casos. Por ejemplo, si la base de conocimiento incluye los axiomas de Peano

$$NumNat(0)$$
$$\forall n \;\; NumNat(n) \Rightarrow NumNat(S(n))$$

entonces el encadenamiento hacia delante añade $NumNat(S(0))$, $NumNat(S(S(0)))$, $NumNat(S(S(S(0))))$, etc. Este problema en general es inevitable. Igual que con la lógica de primer orden general, la implicación con cláusulas positivas es semidecidible.

Encadenamiento hacia delante eficiente

El algoritmo de encadenamiento hacia delante de la Figura 9.3 está diseñado más bien para facilitar su comprensión que para que sea eficiente en su ejecución. Hay tres fuentes de complejidad posibles. Primero, el «bucle interno» del algoritmo requiere que se encuentren todos los unificadores posibles de manera que la premisa se unifique con un conjunto adecuado de hechos de la base de conocimiento. A este proceso, a menudo se le denomina **emparejamiento de patrones** y puede ser muy costoso. Segundo, el algoritmo vuelve a comprobar cada regla en cada iteración para ver si sus premisas se satisfacen, incluso cuando se han realizado muy pocas adiciones a la base de conocimiento en cada iteración. Por último, el algoritmo podría generar muchos hechos que son irrelevantes para el objetivo. Vamos a abordar cada uno de estos problemas por separado.

EMPAREJAMIENTO DE PATRONES

Emparejar reglas con los hechos conocidos

El problema de emparejar la premisa de una regla con los hechos de la base de conocimiento podría parecer algo bastante sencillo. Por ejemplo, suponga que queremos aplicar la regla

$$Misil(x) \Rightarrow Arma(x)$$

Entonces necesitamos encontrar todos los hechos que se unifican con $Misil(x)$; en una base de conocimiento indexada de forma adecuada, esto se puede realizar en tiempo constante por cada hecho. Ahora considere una regla como

$$Misil(x) \wedge Tiene(Nono, x) \Rightarrow Vende(West, x, Nono).$$

Otra vez, podemos encontrar todos los objetos que tiene Nono en un tiempo constante por objeto; entonces, por cada objeto, podríamos comprobar si es un misil. Sin embargo, si la base de conocimiento tiene muchos objetos en posesión de Nono y unos pocos misiles, sería mejor encontrar todos los misiles primero y entonces comprobar si son de Nono. Este es el problema de la **ordenación de los conjuntores**: encontrar una ordenación para resolver los conjuntores de las premisas de una regla de tal manera que el coste se minimice. Resulta que encontrar la ordenación óptima es un problema NP-duro, pero hay buenas heurísticas disponibles. Por ejemplo, la heurística de la **variable más**

ORDENACIÓN DE CONJUNTORES

restringida que se utiliza en los PSR del Capítulo 5 sugeriría ordenar los conjuntores para ver los misiles antes si hay menos misiles que objetos que pertenezcan a Nono.

La conexión entre el emparejamiento de patrones y la satisfacción de restricciones realmente es muy estrecha. Podemos ver cada conjuntor como una restricción sobre las variables que contiene, por ejemplo, *Misil*(x) es una restricción unaria sobre la variable x. Ampliando esta idea, *podemos expresar cada PSR de dominio finito como una única cláusula positiva junto a algunos hechos base asociados a ella*. Considere el problema del coloreado de un mapa de la Figura 5.1 que se muestra en la Figura 9.5(a). Una formulación equivalente mediante una cláusula positiva se muestra en la Figura 9.5(b). Está claro que la conclusión *Coloreable*( ) se puede inferir sólo si el PSR tiene una solución. Ya que los PSR en general incluyen a los problemas 3SAT como un caso especial, podemos concluir que *emparejar una cláusula positiva con un conjunto de hechos es un problema NP-duro*.

Podría parecer algo depresivo que el encadenamiento hacia delante tenga un problema de emparejamiento NP-duro en su bucle interno. Sin embargo, hay tres motivos por los cuales animarse:

COMPLEJIDAD DE DATOS

- Podemos recordarnos a nosotros mismos que muchas reglas en bases de conocimiento del mundo real son más bien pequeñas y sencillas (como las reglas de nuestro problema del crimen) que grandes y complejas (como las de la formulación del PSR de la Figura 9.5). Es algo común asumir, en las bases de datos sobre el mundo, que tanto el tamaño de las reglas como las aridades de los predicados están limitados por una constante y preocuparnos sólo por la **complejidad de los datos**, es decir, la complejidad de la inferencia está en función del número de hechos base en la base de datos. Es fácil demostrar que la complejidad de los datos del encadenamiento hacia delante es polinómica.

(a)

$$Dif(wa, nt) \wedge Dif(wa, sa) \wedge$$
$$Dif(nt, q)\ Dif(nt, sa) \wedge$$
$$Dif(q, nsw) \wedge Dif(q, sa) \wedge$$
$$Dif(nsw, v) \wedge Dif(nsw, sa) \wedge$$
$$Dif(v, sa) \Rightarrow Coloreable()$$

$$Dif(Rojo, Azul)\quad Dif(Rojo, Verde)$$
$$Dif(Verde, Rojo)\ Dif(Verde, Azul)$$
$$Dif(Azul, Rojo)\quad Dif(Azul, Verde)$$

(b)

Figura 9.5 (a) Un grafo de restricciones para colorear el mapa de Australia (el de la Figura 5.1). (b) El PSR del coloreado del mapa como una sola cláusula positiva. Fíjese en que los dominios de las variables se definen implícitamente por las constantes dadas en los hechos base para *Dif*.

- Podemos considerar subclases de reglas para las cuales el emparejamiento sea eficiente. Esencialmente, cada cláusula Datalog se puede ver como la definición de un PSR, así que el emparejamiento será tratable sólo cuando el correspondiente PSR lo sea. El Capítulo 5 describe varias familias tratables de PSR. Por ejemplo, si el grafo de restricciones (el grafo cuyos nodos son las variables y cuyos arcos son las restricciones) forma un árbol, entonces el PSR se puede resolver en tiempo lineal. El mismo resultado se obtiene exactamente con el emparejamiento de reglas. Por ejemplo, si eliminamos Sur de Australia del mapa de la Figura 9.5, la clásusula resultante será

$$Dif(wa, nt) \wedge Dif(nt, q) \wedge Dif(q, nsw) \wedge Dif(nsw, v) \Rightarrow Coloreable(\ )$$

  Que se corresponde con el PSR reducido de la Figura 5.11. Los algoritmos para resolver los PSR con estructura de árbol se pueden aplicar de forma directa al problema de emparejamiento de reglas.
- Podemos trabajar duro para eliminar los intentos de emparejamiento de reglas redundantes en el encadenamiento hacia delante, que es el tema de la siguiente sección.

Encadenamiento hacia delante incremental

Cuando mostramos cómo trabaja el encadenamiento hacia delante mediante el ejemplo del crimen hicimos trampa; en concreto, omitimos alguno de los emparejamiento de reglas que se muestran en el algoritmo de la Figura 9.3. Por ejemplo, en la segunda iteración, la regla

$$Misil(x) \Rightarrow Arma(x)$$

se empareja con $Misil(M_1)$ (otra vez), y desde luego, la conclusión $Arma(M_1)$ ya se conoce y por tanto no pasa nada. Este tipo de emparejamientos de reglas redundantes se puede evitar si hacemos la siguiente observación: *cada hecho nuevo inferido en una iteración t debe ser derivado de al menos un hecho nuevo inferido en la iteración* $t - 1$. Esto es cierto porque cualquier inferencia que no necesite un hecho nuevo de la iteración $t - 1$ podía haberse realizado ya en la iteración $t - 1$.

Esta observación nos lleva de forma natural a un algoritmo de encadenamiento hacia delante incremental, en el que en cada iteración t comprobamos una regla sólo si su premisa incluye algún conjuntor p_i que se unifique con un hecho p_i' que se haya inferido en la iteración $t - 1$. El paso de emparejamiento de regla fija que p_i se empareje con p_i', pero permite que los otros conjuntores de la regla se emparejen con hechos de cualquier iteración previa. Este algoritmo genera exactamente los mismos hechos en cada iteración que en el algoritmo de la Figura 9.3, pero mucho más eficientemente.

Con una indexación adecuada, es fácil identificar todas las reglas que se pueden disparar por un hecho dado, y en efecto, muchos sistemas reales operan en un modo de «actualización» en donde el encadenamiento hacia delante ocurre en respuesta a cada nuevo hecho que es DICHO al sistema. Las inferencias se generan en cascada a través del conjunto de reglas hasta que se encuentra un punto fijo y entonces, el proceso comienza otra vez con el siguiente hecho nuevo.

Generalmente, sólo una pequeña fracción de las reglas de la base de conocimiento es realmente disparada por la adición de un hecho nuevo. Esto significa que se realiza una gran cantidad de trabajo redundante en construir emparejamientos parciales repetidamente que tienen algunas de sus premisas insatisfechas. Nuestro ejemplo del crimen es más bien demasiado pequeño como para mostrar esto de forma efectiva, pero fíjese en que se construye un emparejamiento parcial en la primera iteración entre la regla

$$Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminal(x)$$

y el hecho *Americano*(*West*). Este emparejamiento parcial entonces se descarta y se reconstruye en la segunda iteración (cuando la regla tiene éxito). Sería mejor retener y completar gradualmente el emparejamiento parcial a medida que llegan nuevos hechos, que descartarlo.

RETE

El algoritmo **rete**[3] fue el primero en aplicarse seriamente a este problema. El algoritmo preprocesa el conjunto de reglas de la base de conocimiento para construir una especie de red de flujo de datos en la que cada nodo es un literal de las premisas de la regla. Las ligaduras de las variables fluyen a través de la red y se eliminan cuando fallan al emparejar un literal. Si dos literales en una regla comparten una variable (por ejemplo, $Vende(x, y, z) \wedge Hostil(z)$ en el ejemplo del crimen) entonces las ligaduras de cada literal se filtran mediante un nodo de igualdad. Una ligadura de variables que se encuentra con un nodo con un literal *n*-ario, como $Vende(x, y, z)$ podría tener que esperar a que se establezcan las ligaduras de las otras variables para que el proceso continúe. En un punto dado, el estado de una red rete captura todos los emparejamientos parciales de las reglas, evitando una gran cantidad de cálculo.

SISTEMAS DE PRODUCCIÓN

Las redes rete, y varias mejoras basadas en ellas, han sido un componente clave de los denominados **sistemas de producción**, que fueron de entre los sistemas de encadenamiento hacia delante los de uso más generalizado[4]. El sistema XCON (originalmente llamado R1, McDermott, 1982) se construyó utilizando una arquitectura de sistema de producción. El XCON contenía varios miles de reglas para el diseño de configuraciones de componentes de computador para los clientes de Digital Equipment Corporation. Fue uno de los primeros éxitos comerciales claros en el campo emergente de los sistemas expertos. Muchos otros sistemas similares han sido construidos utilizando la misma tecnología base, que ha sido implementada en el lenguaje de propósito general OPS-5.

ARQUITECTURAS COGNITIVAS

Los sistemas de producción también son populares en las **arquitecturas cognitivas**, es decir, los modelos sobre el razonamiento humano como el ACT (Anderson, 1983) y el SOAR (Laird *et al*., 1987). En estos sistemas, la «memoria de trabajo» del sistema modela la memoria a corto término, y las reglas forman parte de la memoria a largo término. En cada ciclo de ejecución, las producciones se emparejan con los hechos en la memoria de trabajo. Una producción cuyas condiciones se satisfacen puede añadir o eliminar hechos en la memoria de trabajo. En contraste con la típica situación en una base de datos, a menudo, los sistemas de producción tienen muchas reglas y relativamente pocos hechos. Con la tecnología de emparejamiento adecuadamente optimizada, algunos sistemas modernos pueden operar sobre un millón de reglas en tiempo real.

---

[3] Rete es red en Latín. En inglés, su pronunciación rima con *treaty*.
[4] La palabra **producción** de los **sistemas de producción** denota una regla de condición-acción.

Hechos irrelevantes

La última fuente de ineficiencia en el encadenamiento hacia delante parece ser intrínseco en el enfoque y también surge en la lógica proposicional. (*Véase* Apartado 7.5.) El encadenamiento hacia delante realiza todas las inferencias permitidas basadas en los hechos conocidos, *aunque estos sean irrelevantes respecto al objetivo*. En nuestro ejemplo del crimen, no había reglas capaces de trazar conclusiones irrelevantes, así que la falta de dirección no era un problema. En otros casos (por ejemplo, si tenemos varias reglas que describan los hábitos alimenticios de los americanos y los precios de los misiles), PREGUNTA-EHD-LPO generará muchas conclusiones irrelevantes.

Una forma de evitar trazar conclusiones irrelevantes es utilizar el encadenamiento hacia atrás, tal como describiremos en el Apartado 9.4. Otra solución es restringir el encadenamiento hacia delante a un subconjunto seleccionado de las reglas; este enfoque se discutió en el contexto proposicional. Un tercer enfoque ha surgido de la comunidad de las bases de datos deductivas, en las que el encadenamiento hacia delante es la herramienta estándar. La idea es rescribir el conjunto de reglas, utilizando información del objetivo, de tal manera que sólo se consideran las ligaduras de variables relevantes (aquellas que pertenecen al denominado **conjunto mágico**) durante la inferencia hacia delante. Por ejemplo, si el objetivo es *Criminal*(*West*), la regla que concluye *Criminal*(*x*) será rescrita para incluir un conjuntor extra que restringe el valor de la *x*:

CONJUNTO MÁGICO

$$Mágico(x) \wedge Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminal(x)$$

El hecho *Mágico*(*West*) también se añade a la BC. De esta manera, incluso si la base de conocimiento contiene hechos acerca de millones de americanos, sólo se considerará el Coronel West durante el proceso de inferencia hacia delante. El proceso completo para definir los conjuntos mágicos y rescribir la base de conocimiento es demasiado complejo para incluirlo aquí, pero la idea básica es realizar una especie de inferencia hacia atrás «genérica» desde el objetivo para resolver qué ligaduras de las variables necesitan ser restringidas. Por lo tanto, se puede pensar en el enfoque del conjunto mágico como en un tipo híbrido entre la inferencia hacia delante y el preprocesamiento hacia atrás.

9.4 Encadenamiento hacia atrás

La segunda gran familia de algoritmos de inferencia lógica utiliza el enfoque de **encadenamiento hacia atrás** que se introdujo en el Apartado 7.5. Estos algoritmos trabajan hacia atrás desde el objetivo, encadenando a través de las reglas hasta encontrar los hechos conocidos que soportan la demostración. Describiremos el algoritmo básico, y entonces describiremos cómo se utiliza en la **programación lógica**, que es la forma más ampliamente utilizada de razonamiento automático. También veremos que el encadenamiento hacia atrás presenta algunas desventajas comparado con el encadenamiento hacia delante, y veremos mecanismos para vencerlas. Por último, veremos la estrecha conexión entre la programación lógica y los problemas de satisfacción de restricciones.

Un algoritmo de encadenamiento hacia atrás

La Figura 9.6 muestra un algoritmo sencillo de encadenamiento hacia atrás, el PREGUNTA-EHA-LPO. El algoritmo se invoca con una lista de objetivos que contiene un solo elemento, la petición original, y devuelve el conjunto de todas las sustituciones que satisfacen la petición. La lista de objetivos se puede ver como una «pila» a la espera de ser procesada; si *todos* los objetivos se pueden satisfacer, entonces la rama actual de la demostración tiene éxito. El algoritmo toma el primer objetivo de la lista y encuentra cada cláusula de la base de conocimiento cuyo literal positivo, o **cabeza**, se unifica con él. Cada una de estas cláusulas crea una nueva llamada recursiva en la que las premisas, o **cuerpo**, de la cláusula se añaden a la pila de objetivos. Recuerda que los hechos son cláusulas con cabeza y sin cuerpo, así, cuando el objetivo se unifica con un hecho conocido, no se añaden más subobjetivos a la pila, y el objetivo se resuelve. La Figura 9.7 es el árbol de demostración para derivar *Criminal*(*West*) a partir de las sentencias (9.3) hasta la (9.10).

COMPOSICIÓN

El algoritmo utiliza la **composición** de sustituciones. COMPON($\theta 1$, $\theta 2$) es la sustitución cuyo efecto es idéntico a aplicar cada sustitución en orden. Es decir,

$$\text{SUST}(\text{COMPON}(\theta_1, \theta_2), p) = \text{SUST}(\theta_2, \text{SUST}(\theta_1, p))$$

En el algoritmo, las ligaduras de variables actuales, que se almacenan en θ, están compuestas por las ligaduras que resultan de unificar el objetivo con la cabeza de la cláusula, y dan un conjunto nuevo de ligaduras para la siguiente llamada recursiva de la función.

El encadenamiento hacia atrás, tal como lo hemos escrito, es claramente un algoritmo de búsqueda de primero en profundidad. Esto significa que sus requerimientos de espacio son lineales respecto al tamaño de la demostración (olvidando por ahora, el espacio requerido para acumular la solución). También significa que el encadenamiento hacia atrás (a diferencia del encadenamiento hacia delante) sufre algunos problemas con

```
función PREGUNTA-EHA-LPO(BC, objetivos, θ) devuelve un conjunto de sustituciones
  entradas: BC, una base de conocimiento
            objetivos, una lista de conjuntores que forman la petición (θ ya aplicada)
            θ, la sustitución actual, inicialmente la sustitución vacía { }
  variables locales: respuestas, un conjunto de sustituciones, inicialmente vacío

  si objetivos está vacío entonces devolver {θ}
  q′ ← SUST(θ, PRIMERO(objetivos))
  para cada sentencia r en BC hacer donde ESTANDARIZAR-VAR(r) = (p1 ∧ … ∧ pn ⇒ q)
            y θ′ ← UNIFICA(q, q′) tiene éxito
    nuevos_objetivos ← [p1, …, pn | RESTO(objetivos)]
    respuestas ← PREGUNTA-EHA-LPO(BC, nuevos_objetivos, COMPON(θ′, θ)) ∪ respuestas
  devolver respuestas
```

Figura 9.6 Un algoritmo sencillo de encadenamiento hacia atrás.

Figura 9.7. Árbol de prueba mediante encadenamiento hacia atrás para demostrar que West es un criminal. El árbol debería ser leído primero en profundidad, de izquierda a derecha. Para probar *Criminal*(*West*), hay que probar las cuatro conjunciones bajo él. Algunas de ellas están en la base de conocimiento y otras requieren más encadenamientos hacia atrás. Las resoluciones para cada unificación con éxito se muestran después de la submeta correspondiente. Observe que, una vez que una submeta tiene éxito en una conjunción, se aplica su sustitución a las submetas subsecuentes. Así, para cuando PREGUNTA-EHA-LPO llega a la última conjunción, originalmente *Hostil*(*z*), *z* ya está ligada a *Nono*.

los estados repetidos y la incompletitud. Discutiremos estos problemas y algunas soluciones potenciales, pero primero veremos cómo se utiliza el encadenamiento hacia atrás en los sistemas de programación lógica.

Programación lógica

La programación lógica es una tecnología que está bastante relacionada con abarcar el ideal declarativo descrito en el Capítulo 7: dichos sistemas se construirían expresando el conocimiento en un lenguaje formal y los problemas se resolverían ejecutando procesos de inferencia sobre dicho conocimiento. El ideal se resume en la ecuación de Robert Kowalski,

$$Algoritmo = Lógica + Control$$

PROLOG

Prolog es, de lejos, el lenguaje de programación lógica más extensamente utilizado. Sus usuarios se cuentan en cientos de miles. Se utilizó inicialmente como un lenguaje de prototipado rápido y para las tareas de manipulación de símbolos, como la escritura de compiladores (Van Roy, 1990), y en el análisis sintáctico del lenguaje natural (Pereira y Warren, 1980). Muchos sistemas expertos se han escrito en Prolog, para dominios legales, médicos, financieros y muchos otros.

Los programas en Prolog son conjuntos de cláusulas positivas escritas en una notación algo diferente a la estándar de la lógica de primer orden. Prolog utiliza las letras mayúsculas en las variables y las minúsculas en las constantes. Las cláusulas están escritas

con la cabeza precediendo al cuerpo; «`:-`» se utiliza para las implicaciones a la izquierda, las comas separan los literales en el cuerpo, y el punto indica el final de la sentencia:

```
criminal(X) :- americano(X), arma(Y), vende(X, Y, Z), hostil(Z)
```

El Prolog incluye una «sintaxis edulcorada» para la notación de listas y la aritmética. Como ejemplo, aquí tenemos un programa en Prolog para `unir(X, Y, Z)`, que tiene éxito si la lista `Z` es el resultado de unir las listas `X` y `Y`:

```
unir([], Y, Y)
unir([A|X], Y, [A|Z]) :- unir(X, Y, Z)
```

En castellano, podemos leer estas cláusulas como (1) unir una lista vacía a una lista `Y` genera la misma lista `Y`, y (2) `[A|Z]` es el resultado de unir `[A|X]` a Y, dado que `Z` es el resultado de unir `X` a `Y`. Esta definición de unir se asemeja bastante a la definición correspondiente en Lisp, pero realmente es mucho más potente. Por ejemplo, podemos realizar la petición `unir(A, B, [1, 2])`: ¿qué dos listas se pueden unir para dar `[1, 2]`? Obtenemos las soluciones hacia atrás

```
A = []     B = [1, 2]
A = [1]     B = [2]
A = [1, 2]  B = [ ]
```

La ejecución de los programas en Prolog se hace mediante el encadenamiento hacia atrás en primero en profundidad, donde las cláusulas se van intentando en el orden en el que se han escrito en la base de conocimiento. Algunos aspectos del Prolog caen fuera de la inferencia lógica estándar:

- Hay un conjunto de funciones incorporadas para la aritmética. Los literales que utilizan estos símbolos de función se «demuestran» mediante la ejecución de código, y no realizando inferencias adicionales. Por ejemplo, el objetivo «`X es 4+3`» tiene éxito con la `X` ligada a 7. Por el otro lado, el objetivo «`5 es X+Y`» falla, ya que las funciones incorporadas no pueden resolver ecuaciones arbitrarias[5].
- Hay predicados incorporados que tienen efectos colaterales cuando se ejecutan. Estos son predicados de entrada y salida, y predicados de `asertar/retractar` para modificar la base de conocimiento. Este tipo de predicados no tienen su homólogo en la lógica y pueden producir algunos efectos confusos, por ejemplo, si los hechos se asertan en una rama del árbol de demostración, éste finalmente falla.
- El Prolog permite un tipo de negación denominada **negación por fallo**. Un objetivo negado `no P` se considera demostrado si el sistema falla al probar `P`. Así, la sentencia

  ```
  vivo(X) :- no muerto(X)
  ```

  se puede leer como «Todo el mundo está vivo si no es probable que esté muerto».
- El Prolog tiene un operador de igualdad, `=`, pero le falta todo el poder de la igualdad lógica. Un objetivo de igualdad tiene éxito si los dos términos son *unificables* y falla de otro modo. Así, `X+Y=2+3` tiene éxito con la ligadura de `X` a `2` y de `Y` a `3`, pero `estrellamatinal=estrellanocturna` falla. (En la lógica clásica, la

[5] Fíjese en que si se proporcionaran los axiomas de Peano, este tipo de objetivos se podrían resolver mediante inferencia en un programa en Prolog.

última igualdad podría ser o no ser verdadera.) No se pueden asertar hechos o reglas acerca de la igualdad.

- La **comprobación de ocurrencias** se omite en el algoritmo de unificación del Prolog. Esto significa que se pueden realizar algunas inferencias no sólidas: éstas son rara vez un problema excepto cuando se utiliza el Prolog en la demostración de teoremas matemáticos.

Las decisiones tomadas en el diseño del Prolog representan un compromiso entre la eficiencia en la ejecución y el ideal declarativo; en tanto se entendía la eficiencia en el momento en que el Prolog se diseñó. Volveremos a este tema más tarde, al ver cómo se implementó el Prolog.

Implementación eficiente de programas lógicos

La ejecución de un programa en Prolog se puede realizar en dos modos: interpretado o compilado. La interpretación «esencialmente» acumula la ejecución del algoritmo PREGUNTA-EHA-LPO de la Figura 9.6, con el programa como la base de conocimiento. Decimos «esencialmente» porque los intérpretes de Prolog contienen una variedad de mejoras diseñadas para maximizar la velocidad. Aquí sólo consideramos dos.

Primero, en vez de construir la lista de todas las respuestas posibles para cada subobjetivo antes de continuar con el siguiente, los intérpretes de Prolog generan una respuesta y una «esperanza» de generar el resto cuando la respuesta actual haya sido totalmente explorada. Esta esperanza se denomina **punto de elección**. Cuando una búsqueda de primero en profundidad completa su exploración de las soluciones posibles que surgen de la respuesta actual y dan marcha atrás al punto de elección, el punto de elección se expande para tratar la nueva respuesta para el subobjetivo y el nuevo punto de elección. Este enfoque ahorra tiempo y espacio. También proporciona una interfaz muy sencilla para la depuración ya que cada vez sólo hay un único camino solución en consideración.

PUNTO DE ELECCIÓN

Segundo, nuestra sencilla implementación del PREGUNTA-EHA-LPO pierde una gran cantidad de tiempo en generar y componer las sustituciones. El Prolog implementa las

```
procedimiento UNIR(ax, y, az, continuación)

  pista ← PUNTERO-GLOBAL-PISTA()
  si ax = [] y UNIFICA(y, az) entonces LLAMA(continuación)
  REINICIALIZA-PISTA(pista)
  a ← VARIABLE-NUEVA()
  x ← VARIABLE-NUEVA()
  z ← VARIABLE-NUEVA()
  si UNIFICA(ax, [a – x]) y UNIFICA(az, [a – z]) entonces UNIR(x, y, z, continuación)
```

Figura 9.8 Pseudocódigo que representa el resultado de compilar el predicado `Unir`. La función VARIABLE-NUEVA devuelve una variable nueva, distinta de todas las otras variables utilizadas hasta el momento. El procedimiento LLAMA(*continuación*) continúa la ejecución con la continuación especificada.

sustituciones utilizando variables lógicas de las que se pueden recordar sus ligaduras actuales. En cualquier instante de tiempo, todas las variables o no están ligadas, o están ligadas a algún valor. Juntas, estas variables y valores definen implícitamente la sustitución para la rama actual de la demostración. Extender el camino sólo puede añadir nuevas ligaduras de variables, porque un intento de añadir una ligadura distinta para una variable ya ligada generaría un fallo en la unificación. Cuando un camino en la búsqueda falla, el Prolog vuelve al punto de elección previo, y entonces podría tener que desligar algunas variables. Esto se hace guardando la pista de todas las variables que han sido ligadas en una pila llamada **pista**. Como cada nueva variable se liga mediante UNIFICA-VAR, la variable se coloca en la pila. Cuando el objetivo falla y es el momento de volver al punto de elección previo, cada una de las variables se desliga siendo eliminada de la pila.

PISTA

Incluso los intérpretes de Prolog más eficientes requieren de varios miles de instrucciones de máquina por cada paso de inferencia debido al coste del índice de consulta, la unificación, y la construcción de la pila de las llamadas recursivas. En efecto, el intérprete siempre se comporta como si nunca hubiera visto el programa antes; por ejemplo, tiene que *encontrar* las cláusulas que emparejan con el objetivo. Por el otro lado, un programa compilado de Prolog es un procedimiento de inferencia para un conjunto específico de cláusulas, así que *se conoce* qué cláusulas emparejan con el objetivo. El Prolog básicamente genera un demostrador de teoremas en miniatura, y para ello, elimina mucho de los costes de la interpretación. También es posible **abrir-el-código** de la rutina de unificación en cada diferente llamada, y entonces, evitar el análisis explícito de la estructura de los términos. (Para más detalles sobre codificación abierta de la unificación, *véase* Warren *et al.*, (1977).)

CÓDIGO ABIERTO

El conjunto de instrucciones de los computadores actuales nos dan una similitud muy pobre con la semántica del Prolog, así que muchos compiladores de Prolog compilan a un lenguaje intermedio en vez de directamente al lenguaje máquina. El lenguaje intermedio más popular es el *Warren Abstract Machine*, o WAM, nombrado así gracias a David H. D. Warren, uno de los implementadores del primer compilador de Prolog. El WAM es un conjunto abstracto de instrucciones que es apropiado para el Prolog y se puede interpretar o traducir a lenguaje máquina. Otros compiladores traducen el Prolog a un lenguaje de alto nivel, como Lisp o C, y entonces utilizan el compilador de ese lenguaje para traducirlo al lenguaje máquina. Por ejemplo, la definición del predicado `Unir` se puede compilar en el código que se muestra en la Figura 9.8. Hay varios temas que vale la pena tener en cuenta:

- Mejor que tener que buscar las cláusulas `Unir` en la base de conocimiento, las cláusulas se pueden convertir en procedimientos y entonces las inferencias se llevan a cabo simplemente llamando al procedimiento.
- Tal como hemos descrito antes, las ligaduras de las variables se mantienen en la pila. El primer paso del procedimiento guarda el estado actual de la pila, y así se puede restaurar mediante REINICIALIZA-PILA si la cláusula falla. Esto deshace las ligaduras generadas por la primera llamada a UNIFICA.
- El truco consiste en el uso de las **continuaciones** para implementar los puntos de elección. Se puede pensar en una continuación como en el empaquetamiento de un procedimiento y una lista de argumentos que juntos definen lo que se debe ha-

CONTINUACIONES

cer si el objetivo actual ha tenido éxito. Esto no ocurriría con un procedimiento como UNIR cuando tuviera éxito el objetivo, porque podría tenerlo por diversos caminos, y cada uno de ellos debería ser explorado. El argumento de la continuación resuelve este problema porque puede ser llamado cada vez que el objetivo tiene éxito. En el código de UNIR, si el primer argumento está vacío, entonces el predicado UNIR ha tenido éxito. Entonces LLAMAMOS a la continuación con las ligaduras adecuadas en la pila, para hacer seguidamente lo que se debería hacer. Por ejemplo, si la llamada a UNIR fuera en el nivel superior, la continuación imprimiría las ligaduras de las variables.

Antes del trabajo de Warren sobre compilación en Prolog, la programación lógica era demasiado lenta para un uso general. Los compiladores de Warren y otros códigos de Prolog permiten alcanzar velocidades que son competitivas con C en una variedad de puntos de referencia estándar (Van Roy, 1990). Por supuesto, el hecho de que uno pueda escribir un planificador o un analizador sintáctico de lenguaje natural en unas pocas docenas de líneas de código en Prolog lo hace algo más deseable que el prototipado en C, principalmente para proyectos de investigación en IA de escala pequeña.

PARALELISMO-O

PARALELISMO-Y

La paralelización también puede proporcionar una aceleración sustancial. Hay dos fuentes principales de paralelismo. La primera, denominada **paralelismo-O**, viene de la posibilidad de unificar el objetivo con muchas cláusulas distintas de la base de conocimiento. Esto nos da una rama independiente en el espacio de búsqueda que nos puede llevar a una solución potencial, y todas estas ramas se pueden resolver en paralelo. La segunda, denominada **paralelismo-Y**, viene de la posibilidad de resolver en paralelo cada conjuntor del cuerpo de una implicación. El paralelismo-Y es más difícil de alcanzar, porque la solución global para la conjunción requiere ligaduras consistentes para todas las variables. Cada rama conjuntiva debe comunicarse con las otras ramas para asegurar una solución global.

Inferencia redundante y bucles infinitos

Ahora volvemos al talón de Aquiles del Prolog: la desunión entre la búsqueda de primero en profundidad y los árboles de búsqueda que incluyen estados repetidos y caminos infinitos. Considere el siguiente programa lógico, que decide si existe un camino entre dos nodos de un grafo dirigido:

```
camino(X, Z) :- enlace(X, Z)
camino(X, Z) :- enlace(X, Y), enlace(Y, Z)
```

En la Figura 9.9(a) se muestra un simple grafo de tres nodos, descrito por los hechos `enlace(a, b)` y `enlace(b, c)`. Con este programa, la petición `camino(a, c)` genera la demostración que se muestra en la Figura 9.10(a). Por el otro lado, si ponemos las dos cláusulas en el orden

```
camino(X, Z) :- enlace(X, Y), enlace(Y, Z)
camino(X, Z) :- enlace(X, Z)
```

el Prolog seguirá un camino infinito como el que se ve en la Figura 9.10(b). Por lo tanto, el Prolog es **incompleto** como demostrador de teoremas con cláusulas positivas (incluso

Figura 9.9 (a) Encontrar el camino de A a C puede llevar al Prolog a un bucle infinito. (b) Un grafo en el que cada nodo está conectado a dos sucesores de la siguiente capa seleccionados aleatoriamente. Encontrar un camino de A_1 a J_4 requiere 877 inferencias.

Figura 9.10 (a) La demostración de que existe un camino de A a C. (b) Un árbol de demostración infinito generado cuando las cláusulas están en el orden «equivocado».

con los programas Datalog, como muestra este ejemplo) debido a que para algunas bases de conocimiento falla al demostrar sentencias que están implicadas. Fíjese que el encadenamiento hacia delante no sufre de este problema: una vez `camino(a,b)`, `camino(b,c)` y `camino(a,c)` son inferidas, el encadenamiento hacia delante finaliza con éxito.

El encadenamiento hacia atrás en primero en profundidad también tiene problemas con los cálculos redundantes. Por ejemplo, cuando se busca un camino de A_1 a J_4, como se muestra en la Figura 9.9(b), el Prolog realiza 877 inferencias, muchas de las cuales consisten en encontrar todos los caminos posibles a nodos a los que el objetivo no puede enlazarse. Esto es similar al problema de los estados repetidos que se habló en el Capítulo 3. La cantidad total de inferencias puede ser exponencial respecto al número de hechos base que son generados. Si en vez de esto aplicamos el encadenamiento hacia delante, como mucho se pueden generar n^2 hechos `camino(X,Y)` enlazando n nodos. Para el problema de la Figura 9.9(b) sólo se necesitan 62 inferencias.

PROGRAMACIÓN DINÁMICA

El encadenamiento hacia delante aplicado en problemas de búsqueda sobre grafos es un ejemplo de la **programación dinámica**, en la que las soluciones de los subproblemas se van construyendo incrementalmente a partir de subproblemas más pequeños y se guardan para evitar posteriores recálculos. Obtenemos un efecto similar en el sistema de encadenamiento hacia atrás utilizando la **memorización**, es decir, guardando las soluciones de los subobjetivos a medida que se encuentran y entonces se reutilizan aquellas soluciones cuando el subobjteivo se repite, en vez de repetir el cálculo ya realizado. Este es el enfoque que se utiliza en los sistemas de **programación lógica basada en tablas**, que utilizan unos mecanismos de almacenamiento y recuperación eficientes para realizar la memorización. La programación lógica basada en tablas combina la orientación al objetivo del encadenamiento hacia atrás con la eficiencia de la programación dinámica del encadenamiento hacia delante. Y además es un proceso completo para los programas Datalog, lo que significa que el programador no necesita preocuparse más acerca de los bucles infinitos.

PROGRAMACIÓN LÓGICA BASADA EN TABLAS

Programación lógica con restricciones

En nuestra exposición acerca del encadenamiento hacia delante (Apartado 9.3) mostramos cómo se podían codificar los problemas de satisfacción de restricciones (PSR) mediante cláusulas positivas. El Prolog estándar resuelve este tipo de problemas exactamente de la misma manera como lo hace el algoritmo con *backtracking* dado en la Figura 5.3.

Ya que el *backtracking* (vuelta hacia atrás) enumera los dominios de las variables, sólo trabaja con PSRs con **dominio finito**. En términos de Prolog, debe haber un número finito de soluciones para cada objetivo con las variables desligadas. (Por ejemplo, el objetivo `dif(q, sa)`, que dice que Queensland y el Sur de Australia deben tener colores diferentes, tiene seis soluciones si se permiten tres colores.) Los PSRs con dominios infinitos (por ejemplo con variables enteras o reales) requieren algoritmos bastante diferentes, tales como la propagación de ligaduras o la programación lineal.

La siguiente cláusula tiene éxito si tres números satisfacen la desigualdad del triángulo:

```
triangulo(X, Y, Z) :-
      X >= 0, Y >= 0, Z >= 0, X + Y >= Z, Y + Z >= X, X + Z >= Y
```

Si le hacemos al Prolog la petición `triangulo(3, 4, 5)` trabaja bien. Por el otro lado, si preguntamos por `triangulo(3, 4, Z)`, no se encontrará ninguna solución porque el subobjetivo `Z >= 0` no puede ser manejado por el Prolog. La dificultad es que en Prolog las variables deben estar en uno o dos estados: ligadas o desligadas a un término en particular.

PROGRAMACIÓN LÓGICA CON RESTRICCIONES

Ligar una variable a un término concreto se puede ver como un tipo extremo de restricción, a saber, una restricción de igualdad. La **programación lógica con restricciones** (PLR) permite que las variables estén más bien *restringidas* que *ligadas*. Una solución para un programa lógico con restricciones es el conjunto más específico de restricciones en las variables de la petición que se pueden derivar de la base de conocimiento. Por ejemplo, la solución a la petición de `triangulo (3, 4, Z)` es la restricción

`7 >= Z >= 1`. Los programas lógicos con restricciones son tan sólo un caso especial de la PLR en los que las restricciones solución deben ser restricciones de igualdad, o ligaduras.

Los sistemas de PLR incorporan diversos algoritmos de resolución de restricciones para las restricciones que se permiten en el lenguaje. Por ejemplo, un sistema que permite desigualdades lineales en variables reales podría incluir un algoritmo de programación lineal para resolver dichas restricciones. Los sistemas de PLR también pueden adoptar un enfoque mucho más flexible para resolver las peticiones de la programación lógica estándar. Por ejemplo, en vez de primero en profundidad o *backtracking* (vuelta hacia atrás) de izquierda a derecha, podrían utilizar uno de los algoritmos, más eficientes, que se mostraron en el Capítulo 5, incluyendo las heurísticas de ordenación de los conjuntores, salto atrás, condicionamiento del corte del conjunto, etc. Por lo tanto, los sistemas de PLR combinan elementos de los algoritmos de satisfacción de restricciones, de la programación lógica, y de las bases de datos deductivas.

Los sistemas de PLR también pueden tomar ventaja de la variedad de optimizaciones de búsqueda descritas en el Capítulo 5, tales como la ordenación de valores y variables, la comprobación hacia delante y el *backtracking* inteligente. Se han definido una gran diversidad de sistemas que permiten al programador un mayor control sobre el orden de búsqueda en la ejecución de la inferencia. Por ejemplo, el lenguaje MRS (Genesereth y Smith, 1981; Russell, 1985) le permite al programador escribir **meta reglas** para determinar qué conjuntor se intenta primero. El usuario podría escribir una regla diciendo que el objetivo con menos variables debería intentarse antes o podría escribir reglas específicas del dominio para predicados concretos.

META REGLAS

9.5 Resolución

La última de nuestras tres familias de sistemas lógicos está basada en la **resolución**. En el Capítulo 7 vimos que la resolución proposicional es un procedimiento de inferencia mediante refutación que es completo para la lógica proposicional. En esta sección veremos cómo ampliar la resolución a la lógica de primer orden.

El tema de la existencia de procedimientos de demostración completos concierne directamente a los matemáticos. Si se puede encontrar un procedimiento de demostración completo para los enunciados matemáticos, pueden suceder dos cosas: primero, todas las conjeturas se pueden establecer mecánicamente; segundo, todas las matemáticas se pueden establecer como la consecuencia lógica de un conjunto de axiomas fundamentales. El tema de la completitud por lo tanto ha generado algunos de los trabajos matemáticos más importantes del siglo XX. En 1930, el matemático alemán Kurt Gödel demostró el primer **teorema de la completitud** para la lógica de primer orden, mostrando que una sentencia implicada tiene una demostración finita. (Realmente no se encontró un procedimiento *práctico* de demostración hasta que J. A. Robinson publicó su algoritmo de resolución en 1965.) En 1931, Gödel demostró el aún más famoso **teorema de la incompletitud**. El teorema muestra que un sistema lógico que incluye el principio de la inducción (sin el cual muy pocas de las matemáticas discretas se pueden construir) es

TEOREMA DE LA COMPLETITUD

TEOREMA DE LA INCOMPLETITUD

necesariamente incompleto. De aquí, que haya sentencias implicadas, pero no haya disponible una demostración finita en el sistema. La aguja puede estar en un pajar metafórico, pero ningún procedimiento puede garantizar que será encontrada.

A pesar del teorema de Gödel, los demostradores de teoremas basados en la resolución han sido utilizados para derivar teoremas matemáticos, incluyendo varios de los que no se conocían, previamente una demostración. Los demostradores de teoremas también han sido utilizados para verificar diseños de *hardware* y para generar programas correctos desde el punto de vista lógico, entre otras aplicaciones.

Formas normales conjuntivas en lógica de primer orden

Como en el caso proposicional, la resolución en primer orden requiere que las sentencias estén en la **forma normal conjuntiva** (FNC), es decir, una conjunción de cláusulas, donde cada cláusula es una disyunción de literales[6]. Los literales pueden contener variables, que se asume están cuantificadas universalmente. Por ejemplo, la sentencia

$$\forall x \;\; Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminal(x)$$

se transforma a FNC como

$$\neg Americano(x) \vee \neg Arma(y) \vee \neg Vende(x, y, z) \vee \neg Hostil(z) \vee Criminal(x)$$

Cada sentencia en lógica de primer orden se puede convertir a una sentencia en FNC que es equivalente inferencialmente. En concreto, la sentencia en FNC será *insatisfacible* sólo cuando la sentencia original lo sea, así disponemos de una base para hacer demostraciones mediante contradicción con sentencias en FNC.

El procedimiento para la conversión a la FNC es muy parecido al del caso proposicional que vimos en el Apartado 7.5. La principal diferencia viene de la necesidad de eliminar los cuantificadores universales. Mostraremos el procedimiento transformando la sentencia «Todo el mundo que ama a los animales es amado por alguien», o

$$\forall x \;\; [\forall y \;\; Animal(y) \Rightarrow Ama(x, y)] \Rightarrow [\exists y \;\; Ama(y, x)]$$

Los pasos a seguir son los siguientes:

- **Eliminación de las implicaciones:**

$$\forall x \;\; [\neg\forall y \; \neg Animal(y) \vee Ama(x, y)] \vee [\exists y \;\; Ama(y, x)]$$

- **Anidar las ¬:** además de las reglas usuales para las conectivas negadas, necesitamos las reglas para los cuantificadores negados. De este modo tenemos

$$\begin{array}{lcl} \neg\forall x \;\; p & \text{se convierte en} & \exists x \;\; \neg p \\ \neg\exists x \;\; p & \text{se convierte en} & \forall x \;\; \neg p \end{array}$$

[6] Una cláusula también se puede representar como una implicación con una conjunción de átomos a la izquierda y una disyunción de átomos a la derecha, tal como se muestra en el Ejercicio 7.12. Esta forma, que algunas veces se denomina **forma de Kowalski** cuando se escribe con un símbolo de implicación de derecha a izquierda (Kowalski, 1979b), a menudo es más fácil de leer.

Nuestra sentencia pasa a través de las siguientes transformaciones:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Ama(x, y))] \vee [\exists y\ Ama(y, x)]$$
$$\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Ama(x, y))] \vee [\exists y\ Ama(y, x)]$$
$$\forall x\ [\exists y\ Animal(y) \wedge \neg Ama(x, y))] \vee [\exists y\ Ama(y, x)]$$

Fíjese en que el cuantificador universal ($\forall y$) de la premisa de la implicación se convierte en un cuantificador existencial. La sentencia ahora se lee «Hay algún animal que x no ama, o (si éste no es el caso) alguien ama a x». Está claro que el significado de la sentencia original se mantiene.

- **Estandarizar las variables:** para las sentencias del tipo $(\forall x\ P(x)) \vee (\exists x\ Q(x))$, que utilizan el mismo nombre de variable dos veces, se cambia una de las dos variables. Esto evita la confusión posterior al eliminar los cuantificadores. Así tenemos

$$\forall x\ [\exists y\ Animal(y) \wedge \neg Ama(x, y)] \vee [\exists z\ Ama(z, x)].$$

SKOLEMIZACIÓN

- **Skolemizar:** la **Skolemización** es el proceso de borrar los cuantificadores existenciales mediante su eliminación. En el caso más sencillo, se aplica la regla de la Especificación Existencial del Apartado 9.1: transformar $\exists x\ P(x)$ a $P(A)$, donde A es una constante nueva. Si aplicamos esta regla a nuestra sentencia del ejemplo, obtenemos

$$\forall x\ [Animal(A) \wedge \neg Ama(x, A)] \vee Ama(B, x)$$

en donde se obtiene un significado totalmente erróneo: la sentencia dice que todo el mundo o no puede amar al animal A o que es amado por la entidad B. De hecho, nuestra sentencia original le permite a cada persona no poder amar a un animal diferente o ser amado por otra persona. Por lo tanto, lo que queremos es obtener las entidades de Skolem que dependen de x:

$$\forall x\ [Animal(F(x)) \wedge \neg Ama(x, F(x))] \vee Ama(G(x), x)$$

FUNCIÓN DE SKOLEM

Aquí F y G son **funciones de Skolem**. La regla general es que los argumentos de la función de Skolem dependen de las variables cuantificadas universalmente cuyo ámbito abarca a los cuantificadores existenciales. Igual que con la Especificación Existencial, la sentencia skolemizada es *satisfacible* sólo cuando la sentencia original lo es.

- **Eliminar los cuantificadores universales:** en este punto, todas las variables que quedan deben estar cuantificadas universalmente. Más aún, la sentencia es equivalente a una en la que todos los cuantificadores universales se han desplazado a la izquierda. Por lo tanto podemos eliminar los cuantificadores universales:

$$[Animal(F(x)) \wedge \neg Ama(x, F(x))] \vee Ama(G(x), x)$$

- **Distribuir la $\wedge$ respecto la $\vee$:**

$$[Animal(F(x)) \vee Ama(G(x), x)] \wedge [\neg Ama(x, F(x)) \vee Ama(G(x), x)]$$

Este paso también puede requerir extraer las conjunciones y disyunciones anidadas.

La sentencia ahora está en FNC y está compuesta por dos cláusulas. Es algo ilegible. (Podría ser de ayuda explicar que la función de Skolem $F(x)$ se refiere al animal que potencialmente no ama x, mientras que $G(x)$ se refiere a alguien que podría amar a x.) Afortunadamente, las personas rara vez necesitan ver las sentencias en FNC (el proceso de traducción es muy fácilmente automatizable).

La regla de inferencia de resolución

La regla de la resolución para la lógica de primer orden es simplemente una versión elevada de la regla de resolución proposicional que hemos dado en el Apartado 7.5. Dos cláusulas, que se asume están con las variables estandarizadas y así no comparten ninguna variable, se pueden resolver si contienen literales complementarios. Los literales proposicionales complementarios son complementarios si uno es la negación del otro, los literales en lógica de primer orden son complementarios si uno se *unifica* con la negación del otro. De este modo tenemos

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m_1 \vee \ldots \vee m_n}{\textsc{Sust}(\theta, \ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

donde $\textsc{Unifica}(\ell_i, \neg m_j) = \theta$. Por ejemplo, podemos resolver las dos cláusulas

$$[Animal(F(x)) \vee Ama(G(x), x)] \quad \text{y} \quad [\neg Ama(u, v) \vee \neg Mata(u, v)]$$

eliminando los literales complementarios $Ama(G(x), x)$ y $\neg Ama(u, v)$, con el unificador $\theta = \{u/G(x), v/x\}$, para producir la cláusula **resolvente**

$$[Animal(F(x)) \vee \neg Mata(G(x), x)]$$

RESOLUCIÓN BINARIA

La regla que acabamos de dar es la regla de **resolución binaria**, porque resuelve exactamente dos literales. La regla de resolución binaria por sí misma no nos da un procedimiento de inferencia completo. La regla de resolución general resuelve los subconjuntos de los literales en cada cláusula que es unificable. Un enfoque alternativo es ampliar la **factorización** (la eliminación de los literales redundantes) en el caso de la lógica de primer orden. La factorización proposicional reduce dos literales a uno si son *idénticos*; la factorización de primer orden reduce dos literales a uno si éstos son *unificables*. El unificador debe ser aplicado a la cláusula entera. La combinación de la resolución binaria con la factorización sí es completa.

Demostraciones de ejemplo

La resolución demuestra que $BC \models \alpha$ probando que $BC \wedge \neg\alpha$ es *insatisfacible*, por ejemplo, derivando la cláusula vacía. El enfoque algorítmico es idéntico al caso proposicional, descrito en la Figura 7.12, así que no lo repetiremos aquí. En vez de ello, daremos dos demostraciones de ejemplo. La primera es el ejemplo del crimen del Apartado 9.3. Las sentencias en FNC son:

$\neg Americano(x) \vee \neg Arma(y) \vee \neg Vende(x, y, z) \vee \neg Hostil(z) \vee Criminal(x)$
$\neg Misil(x) \vee \neg Tiene(Nono, x) \vee Vende(West, x, Nono)$
$\neg Enemigo(x, América) \vee Hostil(x)$
$\neg Misil(x) \vee Arma(x)$
$Tiene(Nono, M_1)$. $\quad Misil(M_1)$
$Americano(West)$. $\quad Enemigo(Nono, América)$

También incluimos el objetivo negado, $\neg Criminal(West)$. La demostración por resolución se muestra en la Figura 9.11. Fíjese en la estructura: un comienzo de la «columna» sencillo, con la cláusula objetivo, que se resuelve a través de las cláusulas de la base de conocimiento hasta que se genera la cláusula vacía. Esto es algo característico de la resolución con las bases de conocimiento con cláusulas de Horn. De hecho, las cláusulas a lo largo de la columna se corresponden *exactamente* con los valores consecutivos de las variables de los *objetivos* del algoritmo de encadenamiento hacia atrás de la Figura 9.6. Esto se debe a que siempre escogemos resolver con una cláusula cuyo literal positivo se unifique con el literal más a la izquierda de la cláusula «actual» de la columna; y esto es lo que ocurre exactamente con el encadenamiento hacia atrás. De esta manera, el encadenamiento hacia atrás es un caso especial de la resolución, con una estrategia de control concreta para decidir qué resolución se realiza a continuación.

Nuestro segundo ejemplo hace uso de la Skolemización y trata con cláusulas que no son positivas. Esto genera una estructura de demostración algo más compleja. En lenguaje natural el problema se presenta así:

Todo el mundo que ama a todos los animales es amado por alguien.
Cualquiera que mate a un animal no es amado por nadie.
Jack ama a todos los animales.
Jack o Curiosity mataron a la gata, que se llama Tuna.
¿Mató Curiosity a la gata?

Figura 9.11 Una demostración mediante resolución de que West es un criminal.

Primero expresamos, en lógica de primer orden, las sentencias originales, algún conocimiento de base, y el objetivo G negado:

A. $\forall x\ [\forall x\ Animal(y) \Rightarrow Ama(x, y)] \Rightarrow [\exists y\ Ama(y, x)]$
B. $\forall x\ [\forall y\ Animal(y) \wedge Mata(x, y)] \Rightarrow [\forall z\ \neg Ama(z, x)]$
C. $\forall x\ Animal(x) \Rightarrow Ama(Jack, x)$
D. $Mata(Jack, Tuna) \vee Mata(Curiosity, Tuna)$
E. $Gata(Tuna)$
F. $\forall x\ Gata(x) \Rightarrow Animal(x)$
¬G. $\neg Mata(Curiosity, Tuna)$

Ahora aplicamos el procedimiento de conversión para transformar cada sentencia a FNC:

A1. $Animal(F(x)) \vee Ama(G(x), x)$
A2. $\neg \mathrm{Ama}(x, \mathrm{F}(x)) \vee Ama(\mathrm{G}(x), x)$
B. $\neg Animal(y) \vee \neg Mata(x, y) \vee \neg Ama(z, x)$
C. $\neg Animal(x) \vee Ama(Jack, x)$
D. $Mata(Jack, Tuna) \vee Mata(Curiosity, Tuna)$
E. $Gata(Tuna)$
F. $\neg Gata(x) \vee Animal(x)$
¬G. $\neg Mata(Curiotity, Tuna)$

La demostración mediante resolución de que Curiosity mató a la gata se muestra en la Figura 9.12. En lenguaje natural, la demostración se podría parafrasear como sigue:

> Supongamos que Curiosity no mató a Tuna. Sabemos que lo hizo Jack o Curiosity, por lo tanto tiene que ser Jack. Tuna es una gata y los gatos son animales. Ya que cualquiera que mate a un animal no es amado por nadie, entonces deberíamos pensar que nadie ama a Jack. Por el otro lado, Jack ama a los animales, así que alguien lo ama, y entonces llegamos a una contradicción. De todo ello concluimos que Curiosity mató a la gata.

La demostracióm responde a la pregunta «¿Curiosity mató a la gata?», pero a menudo queremos realizar preguntas más generales, del tipo «¿Quién mató a la gata?». La resolución puede responder a este tipo de preguntas, pero necesita hacer un poco más de esfuerzo para obtener la respuesta. El objetivo es $\exists w\ Mata(w, Tuna)$, que cuando se niega

Figura 9.12 Una demostración mediante resolución de que Curiosity mató a la gata. Fíjese en el uso de la factorización en la derivación de la cláusula $Ama(G(Jack), Jack)$.

queda de la forma ¬*Mata*(*w*, *Tuna*) en FNC. Al repetir la demostración de la Figura 9.12 con el nuevo objetivo negado, obtenemos un árbol de demostración algo similar, pero con la sustitución {*w*/*Curiosity*} en uno de los pasos. Así, en este caso, averiguar quién mató a la gata es justamente una forma de guardar la pista de las ligaduras de la variable de la petición durante la demostración.

DEMOSTRACIÓN NO CONSTRUCTIVA

Desafortunadamente, la resolución puede generar **demostraciones no constructivas** cuando trabaja con objetivos existenciales. Por ejemplo, ¬*Mata*(*w*, *Tuna*) se resuelve con *Mata*(*Jack*, *Tuna*) ∨ *Mata*(*Curiosity*, *Tuna*) para dar *Mata*(*Jack*, *Tuna*), y que se resuelve, otra vez, con ¬*Mata*(*w*, *Tuna*) para obtener la cláusula vacía. Fíjese en que *w* tiene dos ligaduras diferentes en esta demostración; la resolución nos dice que sí, que alguien mató a Tuna: Jack o Curiosity. ¡Esto no nos sorprende! Una solución consiste en restringir los pasos permitidos de la resolución de manera que las variables de la petición se puedan ligar una sola vez en una demostración dada; entonces necesitamos ser capaces de volver sobre las ligaduras posibles. Otra solución es añadir un **literal de respuesta** especial al objetivo negado, que se convierte en ¬*Mata*(*w*, *Tuna*) ∨ *Respuesta*(*w*). Entonces, ahora el proceso de resolución genera una respuesta cuando se genera una cláusula con *sólo el* literal de respuesta. Para la demostración de la Figura 9.12 sería *Respuesta*(*Curiosity*). La demostración no constructiva generaría la cláusulas *Respuesta*(*Curiosity*) ∨ *Respuesta*(*Jack*), que no constituye en sí una respuesta.

LITERAL DE RESUESTA

Completitud de la resolución

En esta sección damos una demostración de la completitud de la resolución. Se puede saltar sin temor por aquellos que están dispuestos a desafiar la fe.

COMPLETITUD DE LA REFUTACIÓN

Demostraremos que la resolución es un procedimiento de **refutación completo**, lo que significa que *si* un conjunto de sentencias es *insatisfacible*, entonces la resolución siempre será capaz de derivar una contradicción. La resolución no se puede utilizar para generar todas las consecuencias lógicas de un conjunto de sentencias, pero se puede utilizar para establecer si una sentencia dada se deduce de un conjunto de sentencias. De aquí, que se pueda utilizar para encontrar todas las respuestas a una pregunta dada, utilizando el método del objetivo negado que se ha descrito previamente en este capítulo.

Tomaremos como premisa que cualquier sentencia en lógica de primer orden (sin igualdad) se puede transformar en un conjunto de cláusulas en FNC. Esto se puede demostrar por inducción sobre la forma de la sentencia, utilizando las sentencias atómicas como el caso base (Davis y Putnam, 1960). Nuestro objetivo, por lo tanto, es demostrar lo siguiente: *si S es un conjunto de cláusulas insatisfacible, entonces la aplicación de un número finito de pasos de resolución sobre S nos dará una contradicción.*

Nuestro esbozo de la demostración se basa en la demostración original de Robinson, con algunas simplificaciones cogidas de Genesereth y Nilsson (1987). La estructura básica de la demostración se muestra en la Figura 9.13; que procede como sigue:

1. Primero, observamos que si *S* es *insatisfacible*, entonces debe existir un subconjunto de *instancias base* de las cláusulas de *S* que también es *insatisfacible* (teorema de Herbrand).

Figura 9.13 Estructura de la demostración de la completitud de la resolución.

2. Entonces apelamos al **teorema fundamental de la resolución** del Capítulo 7, que establece que la resolución proposicional es completa para sentencias base.
3. Luego utilizamos el **lema de la elevación** que muestra que para cualquier demostración mediante resolución proposicioonal que utiliza un conjunto de sentencias base, existe su correspondiente demostración mediante resolución de primer orden que utiliza las sentencias de primer orden de las que se obtuvieron las sentencias base.

Para llevar a cabo el primer paso necesitaremos tres conceptos nuevos:

UNIVERSO DE HERBRAND

- **Universo de Herbrand:** si S es un conjunto de cláusulas, entonces H_s, el universo de Herbrand de S, es el conjunto de todos los términos base que se pueden construir a partir de:

  a) Los símbolos de función en S, si hay alguno.
  b) Los símbolos de constante en S, si hay alguno; sino, el símbolo de constante A.

  Por ejemplo, si S contiene la cláusula $\neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)$, entonces H_s es el siguiente conjunto de términos base:

  $$\{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), \ldots\}.$$

SATURACIÓN

- **Saturación (Instancias Base):** si S es el conjunto de cláusulas, y P es el conjunto de términos base, entonces $P(S)$, la saturación de S respecto a P. Es el conjunto de las cláusulas base que se obtienen de aplicar todas las sustituciones posibles consistentes de términos base en P, sobre las variables de S.

BASE DE HERBRAND

- **Base de Herbrand:** la saturación (o las instancias base) de un conjunto S de cláusulas con respecto a su universo de Herbrand se denomina Base de Herbrand de S, que se escribe $Hs(S)$. Por ejemplo, si S contiene tan sólo la cláusula que hemos dado anteriormente, entonces $Hs(S)$ es el conjunto finito de cláusulas

El teorema de incompletitud de Gödel

Ampliando un poco el lenguaje de la lógica de primer orden para permitir el uso de la **inducción matemática**, Gödel fue capaz de demostrar, en su **teorema de la incompletitud**, que hay sentencias aritméticas que son verdaderas y que no se pueden demostrar.

La demostración del teorema de la incompletitud es algo que va más allá del alcance de este libro, y ocupa, como lo hace, un mínimo de 30 páginas, pero aquí podemos dar una breve pista. Comenzamos con la teoría lógica de los números. En esta teoría, hay una sola constante, el 0, y una sola función, S (la función sucesor). En el modelo deseado, S (0) denota 1, $S(S(0))$ denota 2, etc.; por lo tanto, el lenguaje tiene nombres para todos los números naturales. El vocabulario también incluye los símbolos de función $+$, $\times$, y *Exp* (exponenciación) y el conjunto habitual de conectivas lógicas y cuantificadores. El primer paso consiste en fijarse en que el conjunto de sentencias que podemos escribir en este lenguaje se puede enumerar. (Imagine definir un orden alfabético sobre los símbolos y entonces organizar, por orden alfabético, cada conjunto de sentencias de longitud 1, 2, etc.) Entonces podemos enumerar cada sentencia α con un único número natural $\#\alpha$ (el **número de Gödel**). Esto es crucial: la teoría de los números contiene un nombre para cada una de sus sentencias. De forma similar, podemos enumerar cada posible demostración D con un número de Gödel $G(D)$, porque una demostración simplemente es una secuencia finita de sentencias.

Ahora suponga que tenemos un conjunto A de sentencias que es enumerable recursivamente y que es el conjunto de los enunciados acerca de los números naturales que son verdaderos. Y dado que A se puede nombrar mediante un conjunto de enteros, podemos imaginar escribir en nuestro lenguaje una sentencia $\alpha(j, A)$ de la siguiente manera:

> $\forall i$ i no es el número de Gödel de una demostración de una sentencia cuyo número de Gödel es j, donde la demostración sólo utiliza las premisas de A.

Entonces dejemos que σ sea la sentencia $\alpha(\#\sigma, A)$, es decir, una sentencia que define su propia improbabilidad a partir de A. (Que esta sentencia siempre existe es verdadero, pero no del todo obvio.)

Ahora hacemos el siguiente argumento ingenioso: supongamos que σ es probable a partir de A; entonces σ es falsa (porque σ dice que no se puede demostrar). Pero entonces tenemos una sentencia falsa que es probable a partir de A, y por tanto A no tiene sólo sentencias verdaderas (generando una violación de nuestra premisa). Por lo tanto σ no es probable a partir de A. Y esto es exactamente lo que σ proclama; de aquí que σ sea una sentencia verdadera.

Así hemos demostrado (eliminando 29 páginas y media) que para cualquier conjunto de sentencias verdaderas acerca de la teoría de los números, y en concreto, cualquier conjunto de los axiomas básicos, hay otras sentencias verdaderas que *no se pueden* demostrar a partir de esos axiomas. Esto establece, entre otras cosas, que nunca podemos demostrar todos los teoremas de las matemáticas *a partir de un sistema de axiomas dado*. Está claro que esto fue un importante descubrimiento para los matemáticos. Su utilidad para la IA ha sido ampliamente debatida, comenzando con las especulaciones por el propio Gödel. Retomaremos este debate en el Capítulo 26.

$$\{\neg P(A, F(A, A)) \vee \neg Q(A, A) \vee R(A, B),$$
$$\neg P(B, F(B, A)) \vee \neg Q(B, A) \vee R(B, B),$$
$$\neg P(F(A, A), F(F(A, A), A)) \vee \neg Q(F(A, A), A) \vee R(F(A, A), B),$$
$$\neg P(F(A, B), F(F(A, B), A)) \vee \neg Q(F(A, B), A) \vee R(F(A, B), B), \ldots\}$$

TEOREMA DE HERBRAND

Estas definiciones nos permiten definir una variante del **teorema de Herbrand** (Herbrand, 1930):

> Si un conjunto de cláusulas S es *insatisfacible*, entonces debe existir un subconjunto finito de $Hs(S)$ que también sea *insatisfacible*.

Sea S' el subconjunto finito de sentencias base. Ahora podemos apelar al teorema fundamental de la resolución (Apartado 7.5) para demostrar que el **cierre de la resolución** $CR(S')$ contiene la cláusula vacía. Es decir, ejecutando la resolución proposicional para obtener una conclusión a partir de S' genera una contradicción.

Ahora acabamos de establecer que siempre hay una demostración mediante resolución que involucra a algún subconjunto finito de la base de Herbrand de S, el siguiente paso consistirá en demostrar que hay una demostración mediante resolución utilizando las propias cláusulas de S, que no son necesariamente cláusulas base. Comenzamos teniendo en cuenta una sola aplicación de la regla de la resolución. El lema básico de Robinson implica el siguiente hecho:

> Sean C_1 y C_2 dos cláusulas con variables no compartidas, y sean C_1' y C_2' las instancias base de C_1 y C_2. Si C' es el resolvente de C_1' y C_2' entonces debe existir una cláusula C tal que (1) C sea el resolvente de C_1 y C_2 y (2) C' sea una instancia base de C.

LEMA DE ELEVACIÓN

Esto se llama un **lema de elevación**, porque eleva un paso de demostración a partir de cláusulas base a cláusulas de primer orden generales. Para demostrar su lema de elevación básico, Robinson tenía que inventar la unificación y derivar todas la propiedades de los unificadores más generales. Más que repetir aquí la demostración, simplemente ilustramos el lema:

$$C_1 = \neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)$$
$$C_2 = \neg N(G(y), z) \vee P(H(y), z)$$
$$C_1' = \neg P(H(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B)$$
$$C_2' = \neg N(G(B), F(H(B), A)) \vee P(H(B), F(H(B), A))$$
$$C' = \neg N(G(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B)$$
$$C = \neg N(G(y), F(H(y), A)) \vee \neg Q(H(y), A) \vee R(H(y), B)$$

Vemos que efectivamente, C' es una instancia base de C. Por lo general, para que C_1' y C_2' tengan algún resolvente, deben estar construidos aplicando primero el unificador más general a C_1 y C_2 partiendo de una pareja suya de literales complementarios. Del lema de la elevación, es fácil derivar un enunciado similar acerca de cualquier secuencia de aplicaciones de la regla de la resolución:

> Para cualquier cláusula C' del cierre de la resolución de S' hay una cláusula C en el cierre de la resolución de S, tal que C' es una instancia base de C y la derivación de C es de la misma longitud que la derivación de C'.

De este hecho, se sigue que si aparece la cláusula vacía en el cierre de la resolución de S', también debe aparecer en el cierre de la resolución de S. Esto se debe a que la cláu-

sula vacía no puede ser una instancia base de cualquier otra cláusula. Para recapitular: hemos demostrado que si S es *insatisfacible*, entonces hay una derivación finita de la cláusula vacía mediante la regla de la resolución.

La elevación de la demostración del teorema a partir de las cláusulas base a las cláusulas de primer orden nos proporciona un incremento vasto de poder. Este incremento viene del hecho de que la demostración en primer orden necesita variables instanciadas sólo a medida que es necesario para la demostración, mientras que los métodos con cláusulas base eran necesarios para evaluar un número enorme de instanciaciones arbitrarias.

Manejar la igualdad

Ninguno de los métodos de inferencia descritos hasta ahora manejan la igualdad. Hay tres enfoques diferentes que se pueden tomar. El primer enfoque consiste en axiomatizar la igualdad (anotar las sentencias con la relación de igualdad en la base de conocimiento). Necesitamos decir que la igualdad es reflexiva, simétrica y transitiva, y también necesitamos decir que podemos sustituir objetos iguales en cualquier predicado o función.

Por lo que necesitamos tres axiomas básicos, y también uno para cada predicado y función:

$$
\begin{aligned}
&\forall x \;\; x = x \\
&\forall x, y \;\; x = y \Rightarrow y = x \\
&\forall x, y, z \;\; x = y \wedge y = z \Rightarrow x = x \\
&\forall x, y \;\; x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)) \\
&\forall x, y \;\; x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)) \\
&\quad\vdots \\
&\forall w, x, y, z \;\; w = y \wedge x = z \Rightarrow (F_1(w, x) = F_1(y, z)) \\
&\forall w, x, y, z \;\; w = y \wedge x = z \Rightarrow (F_2(\mathrm{w}, x) = F_2(y, z)) \\
&\quad\vdots
\end{aligned}
$$

Dadas estas sentencias, un procedimiento de inferencia estándar como la resolución puede realizar tareas que requieran razonamiento con igualdad, tales como la resolución de ecuaciones matemáticas.

Otra forma de tratar la igualdad es mediante una regla de inferencia adicional. La regla más sencilla, denominada **demodulación**, toma una cláusula unitaria $x = y$ y sustituye la y por cualquier término que se unifica con x en cualquier otra cláusula. Más formalmente, tendríamos

DEMODULACIÓN

- **Demodulación:** para cualquier término x, y y z, donde UNIFICA(x, z) $= \theta$ y $m_n[z]$ es un literal que contiene a z:

$$\frac{x = y, \qquad m_1 \vee \ldots \vee m_n[z]}{m_1 \vee \ldots \vee m_n[\mathrm{SUST}(\theta, y)]}$$

La demodulación se utiliza por lo general para simplificar expresiones que utilizan grupos de aserciones del tipo $x + 0 = x$, $x^1 = x$, etc. La regla también se puede ampliar para manejar cláusulas no unitarias en las que aparece una igualdad de literales:

PARAMODULACIÓN

- **Paramodulación:** para cualquier término x, y y z, donde UNIFICA(x, z) = θ,

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee x = y, m_1 \vee \ldots \vee m_n[z]}{\text{SUST}(\theta, \ell_1 \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_n[y])}$$

A diferencia de la demodulación, la paramodulación nos lleva a un procedimiento de inferencia completo para la lógica de primer orden con igualdad.

Un tercer enfoque maneja el razonamiento con igualdad insertándolo por entero en un algoritmo de unificación extendida. Es decir, los términos son unificables si son *probablemente* iguales bajo alguna sustitución, donde «probablemente» indica algún tipo de razonamiento con igualdad. Por ejemplo, los términos $1 + 2$ y $2 + 1$ normalmente no son unificables, pero un algoritmo de unificación que supiera que $x + y = y + x$ podría unificarlos con la sustitución vacía. Este tipo de **unificación de ecuaciones** se puede realizar mediante algoritmos eficientes diseñados para los axiomas concretos que se vayan a utilizar (conmutatividad, asociatividad, etc.); más que a través de una inferencia explícita con dichos axiomas. Los demostradores de teoremas que utilizan esta técnica están estrechamente relacionados con los sistemas de programación lógica que se describieron en el Apartado 9.4.

UNIFICACIÓN DE ECUACIONES

Estrategias de resolución

Sabemos que las aplicaciones repetidas de la regla de inferencia de resolución finalmente encontrarán una demostración si ésta existe. En esta subsección, examinaremos las estrategias que ayudan a encontrar demostraciones *eficientemente*.

Resolución unitaria

Esta estrategia prefiere realizar las resoluciones cuando una de las sentencias es un único literal (también conocida como **cláusula unitaria**). La idea en que se basa la estrategia es que estamos intentando producir una cláusula vacía, así que podría ser bueno preferir inferencias que produzcan cláusulas más cortas. Resolviendo una sentencia unitaria (como P) con otra sentencia (como $\neg P \vee \neg Q \vee R$) siempre obtenemos una cláusula (en este caso $\neg Q \vee R$) que es más corta que la anterior. Cuando se intentó por primera vez la estrategia de preferencia unitaria en 1964 se obtuvo a una aceleración considerable del proceso, y se hizo viable demostrar teoremas que no se podían manejar sin dicha preferencia. Sin embargo, la preferencia unitaria por sí misma no reduce lo suficiente el factor de ramificación en problemas de tamaño mediano para hacerlos resolubles mediante la resolución. No obstante, es una heurística útil que se puede combinar con otras estrategias.

RESOLUCIÓN UNITARIA

La **resolución unitaria** es una variante restringida de la resolución, en la que cada paso de resolución debe involucrar a una cláusula unitaria. En general, la resolución unitaria es incompleta, pero es completa con las bases de conocimiento de Horn. Las demostraciones mediante resolución unitaria aplicadas a las bases de conocimiento de Horn se parecen bastante al encadenamiento hacia delante.

Resolución mediante conjunto soporte

Las preferencias que intentan antes ciertas resoluciones son de alguna ayuda, pero por lo general es más efectivo intentar eliminar completamente algunas resoluciones potenciales. La estrategia del conjunto soporte hace justamente eso. Comienza por identificar un subconjunto de sentencias denominado **conjunto soporte**. Cada resolución combina una sentencia del conjunto soporte con otra sentencia y añade el resolvente al conjunto soporte. Si el conjunto soporte es relativamente pequeño respecto a la base de conocimiento, el espacio de búsqueda se reduce drásticamente.

CONJUNTO SOPORTE

Tenemos que ser cautos con este enfoque, porque una elección errónea del conjunto soporte hará que el algoritmo sea incompleto. Sin embargo, si elegimos el conjunto soporte S de tal manera que el resto de sentencias sean conjuntamente *satisfacibles*, entonces la resolución mediante el conjunto soporte será completa. Un enfoque habitual es utilizar una petición negada como el conjunto soporte, bajo la asunción que la base de conocimiento original es consistente. (Después de todo, si es consistente, entonces el hecho de la petición que se sigue es vacuo.) La estrategia del conjunto soporte tiene la ventaja adicional de generar árboles de demostración que a menudo son fáciles de entender por las personas, ya que éstas están orientadas al objetivo.

Resolución lineal

RESOLUCIÓN DE ENTRADA

En la estrategia de **resolución de entrada** cada resolución combina una de las sentencias de entrada (de la *BC* o de la petición) con alguna otra sentencia. La demostración de la Figura 9.11 sólo utiliza resoluciones de entrada y tiene la forma característica de una «espina de pez» con sentencias simples que se van combinando en la columna. En las bases de conocimiento de Horn, el Modus Ponens es un tipo de estrategia de resolución de entrada, porque combina una implicación de la BC original con algunas otras sentencias. Por lo tanto, no nos sorprende que la resolución de entrada sea completa con las bases de conocimiento que están en forma de Horn, pero es incompleta en el caso general. La estrategia de **resolución lineal** es una generalización leve que permite que P y Q se resuelvan juntas aunque P esté en la *BC* original o P sea un padre de Q en el árbol de demostración. La resolución lineal es completa.

RESOLUCIÓN LINEAL

Subsunción

El método de **subsunción** elimina todas las sentencias que están subsumidas por (por ejemplo, son más específicas que) una sentencia existente en la *BC*. Por ejemplo, si $P(x)$ está en la BC, entonces no tiene sentido añadir $P(A)$ y aún menos sentido tiene añadir $P(A) \vee Q(B)$. La subsunción ayuda a mantener la *BC* con un tamaño relativamente pequeño, y de esta manera nos ayuda a establecer que el espacio de búsqueda sea, también, relativamente pequeño.

SUBSUNCIÓN

Demostradores de teoremas

Los demostradores de teoremas (también conocidos como razonadores automáticos) se diferencian de la programación lógica en dos cosas. Primero, muchos lenguajes de programación lógica sólo manejan cláusulas de Horn, mientras que los demostradores de teoremas aceptan la lógica de primer orden en su totalidad. Segundo, los programas en Prolog entrelazan la lógica y el control. La elección del programador sobre `A :- B, C` en vez de `A :- C, B` afecta a la ejecución del programa. En muchos demostradores de teoremas, la forma sintáctica escogida para las sentencias no afecta a los resultados. Los demostradores de teoremas todavía necesitan la información de control para poder operar eficientemente, pero por lo general, dicha información se mantiene separada de la base de conocimiento, en vez de formar parte de la propia representación del conocimiento. Gran parte de la investigación en demostradores de teoremas se basa en encontrar estrategias de control que sean de utilidad general al mismo tiempo que aumenten la velocidad.

Diseño de un demostrador de teoremas

En esta sección describiremos el demostrador de teoremas OTTER (*Organized Tecniques for Theorem-proving and Effective Research*) (McCune, 1992), con una atención especial en su estrategia de control. Al preparar un problema para OTTER el usuario debe dividir el conocimiento en cuatro partes:

- Un conjunto de cláusulas, conocido como el **conjunto soporte** (o *cs*), que define los hechos relevantes acerca del problema. Cada paso de la resolución resuelve un miembro del conjunto soporte frente a otro axioma, de esta manera la búsqueda se focaliza en el conjunto soporte.
- Un conjunto de **axiomas utilizables** que son externos al conjunto soporte. Éstos proporcionan el conocimiento base acerca del dominio del problema. El límite entre lo que forma parte del problema (y por lo tanto del *cs*) y lo que es conocimiento base (y por lo tanto de los axiomas utilizables) se deja a elección del usuario.
- Un conjunto de ecuaciones conocido como **rescritores** o **demoduladores**. Aunque los demoduladores son ecuaciones, siempre se aplican en la dirección izquierda a derecha. Por lo tanto, definen la forma canónica en la que todos los términos serán simplificados. Por ejemplo, el demodulador $x + 0 = x$ dice que cada término del tipo $x + 0$ debe reemplazarse por el término x.
- Un conjunto de parámetros y cláusulas que definen la estrategia de control. En concreto, el usuario especifica una función heurística para controlar la búsqueda, y una función de filtrado para eliminar aquellos subobjetivos que no interesan.

OTTER trabaja resolviendo, de forma continuada, un elemento del conjunto soporte frente a uno de los axiomas utilizables. A diferencia del Prolog, utiliza un tipo de búsqueda de primero el mejor. Su función heurística mide el «peso» de cada cláusula, donde las más ligeras tienen preferencia. La elección exacta de la heurística depende del usuario,

pero por lo general, el peso de una cláusula debería ser correlacionada con su tamaño y dificultad. Las cláusulas unitarias se tratan como ligeras; de este modo, la búsqueda se puede ver como una generalización de la estrategia de preferencia unitaria. En cada paso, OTTER mueve la cláusula «más ligera» del conjunto soporte a una lista, a la que llamamos lista utilizable, y añade a dicha lista las consecuencias inmediatas de resolver la cláusula con los elementos de la lista. OTTER finaliza cuando ha encontrado una refutación o no hay más cláusulas en el conjunto soporte. En la Figura 9.14 se muestra el algoritmo en más detalle.

```
procedimiento OTTER(cs, utilizable)
   entradas: cs, el conjunto soporte: cláusulas que definen el problema (una variable global)
             utilizable, conocimiento base potencialmente relevante para el problema

   repetir
      cláusula ← el miembro más ligero de cs
      mover cláusula de cs a utilizable
      PROCESA(INFIERE(cláusula, utilizable), cs)
   hasta cs = [ ] o se ha encontrado una refutación
------------------------------------------------------------------------------
función INFIERE(cláusula, utilizable) devuelve cláusulas

   resuelve cláusula con cada miembro de utilizable
   devolver las cláusulas resultantes después de aplicar FILTRO
------------------------------------------------------------------------------
procedimiento PROCESA(cláusulas, cs)

   para cada cláusula en cláusulas hacer
      cláusula ← SIMPLIFICA(cláusula)
      fusiona literales idénticos
      descarta la cláusula si es una tautología
      cs ← [cláusula – cs]
      si cláusula no tiene literales entonces se ha encontrado una refutación
      si cláusula tiene un literal entonces averigua si hay refutación unitaria
```

Figura 9.14 Esquema del demostrador de teoremas OTTER. El control heurístico se aplica en la selección de la cláusula «más ligera» y en la función FILTRO, que elimina las cláusulas que no se consideran interesantes.

Ampliar el Prolog

Una alternativa en la construcción de un demostrador de teoremas es comenzar con un compilador de Prolog y ampliarlo para conseguir un razonador sólido y completo para toda la lógica de primer orden. Este era el enfoque que se tomó en el *Prolog Technology Theorem Prover*, o PTTP (Stickel, 1988). El PTTP incluye cinco cam-

bios significativos aplicados al Prolog para conseguir completitud y una mayor expresividad:

- La comprobación de ocurrencias se introduce en la rutina de unificación para hacerla sólida.
- La búsqueda de primero en profundidad se reemplaza por una búsqueda de profundidad iterativa. Esto hace que la estrategia de búsqueda sea completa y que tome sólo un factor constante de tiempo.
- Se permite utilizar literales negados (como $\neg P(x)$). En la implementación, hay dos rutinas separadas, una que intenta demostrar P y otra que intenta demostrar $\neg P$.
- Una cláusula con n átomos se almacena como n reglas diferentes. Por ejemplo, $A \Leftarrow B \wedge C$ también podría almacenarse como $\neg B \Leftarrow C \wedge \neg A$ y como $\neg C \Leftarrow B \wedge \neg A$. Esta técnica, conocida como **bloqueo**, significa que el objetivo actual sólo necesita ser unificado con la cabeza de cada cláusula, y aún permite un manejo adecuado de la negación.

BLOQUEO

- La inferencia se hace completa (aun con cláusulas de Horn) añadiendo la regla de resolución de entrada lineal. Si el objetivo actual se unifica con la negación de uno de los objetivos de la pila, entonces el objetivo se puede considerar resuelto. Ésta es una forma de razonamiento por contradicción. Suponga que el objetivo original es P y esto se reduce a una serie de inferencia con el objetivo $\neg P$. Esto establece que $\neg P \Rightarrow P$, que es equivalente lógicamente a P.

A pesar de estos cambios, PTTP mantiene las características que hacen que el Prolog sea rápido. La unificación todavía se hace modificando las variables directamente, desligándolas mediante el relajamiento de la pila en la vuelta hacia atrás (*backtracking*). La estrategia de búsqueda todavía se basa en la resolución de entrada, entonces cada resolución se realiza a través de una de las cláusulas del enunciado original del problema (más que con una cláusula derivada). Esto hace que sea viable compilar todas las cláusulas del enunciado original del problema.

El principal inconveniente del PTTP es que el usuario tiene que renunciar a todo el control sobre la búsqueda de las soluciones. Cada regla de inferencia se utiliza por el sistema tanto en su forma original como en su forma contrapositiva. Esto nos puede llevar a búsquedas nada intuitivas. Por ejemplo, considere la regla

$$(f(x, y) = f(a, b)) \Leftarrow (x = a) \wedge (y = b)$$

Como regla de Prolog, es algo razonable intentar demostrar que los dos términos de f son iguales. Pero el PTTP también generaría su contrapositiva:

$$(x \neq \mathrm{a}) \Leftarrow (f(x, y) \neq f(a, b)) \wedge (y = b)$$

Lo que parece un derroche el intentar demostrar que los dos términos x y a son diferentes.

Demostradores de teoremas como asistentes

Hasta ahora, hemos enfocado los sistemas de razonamiento como agentes independientes que tienen que tomar decisiones y actuar por sí mismos. Otro uso de los demostradores de teoremas es como asistentes, proporcionando consejo a, por ejemplo, un

matemático. De este modo, un matemático actúa como un supervisor, seleccionando la estrategia para determinar qué hacer seguidamente y pidiendo al demostrador de teoremas que rellene los detalles. Este mecanismo alivia, en parte, el problema de la semidecidibilidad, porque el supervisor puede cancelar una petición e intentar otro enfoque si la petición está consumiendo mucho tiempo. Un demostrador de teoremas también puede actuar como un **comprobador de demostraciones**, donde la demostración la da una persona en forma de un conjunto de pasos algo extenso, y las inferencias concretas que se necesitan para demostrar que cada paso es sólido las realiza el sistema.

COMPROBADOR DE DEMOSTRACIONES

RAZONADOR

Un **razonador Socrático** es un demostrador de teoremas cuya función PREGUNTA es incompleta, pero que siempre puede llegar a una solución si le han dado el orden correcto de peticiones. De este modo, los razonadores Socráticos son buenos asistentes, dado que haya un supervisor que envíe las series de llamadas correctas a PREGUNTA. ONTIC (McAllester, 1989) es un sistema de razonamiento Socrático para matemáticos.

Usos prácticos de los demostradores de teoremas

Los demostradores de teoremas han surgido con resultados matemáticos muy originales. El programa SAM (*Semi-Automated Mathematics*) fue el primero en demostrar un lema de la teoría de retículos (Guard *et al*., 1969). El programa AURA también ha respondido a preguntas abiertas en diversas áreas de las matemáticas (Wos y Winker, 1983), el demostrador de teoremas Boyer-Moore (Boyer y Moore, 1979) se ha utilizado y ampliado durante muchos años y se utilizó por Natarajan Shankar para dar la primera demostración formal rigurosa sobre el Teorema de la Incompletitud de Gödel (Shankar, 1986). El programa OTTER es uno de los mejores demostradores de teoremas; se ha utilizado para resolver diversas preguntas abiertas sobre lógica combinatoria. La más famosa es la que concierne al **álgebra de Robbins**. En 1933, Herbert Robbins propuso un conjunto sencillo de axiomas que intentaban definir el álgebra Booleana, pero no se pudo encontrar ninguna prueba que los demostrara (a pesar del trabajo serio de diversos matemáticos incluyendo al propio Alfred Tarski). En octubre de 1996, después de ocho días de cálculo, el EQP (una variante del OTTER) encontró una demostración (McCune, 1997).

ÁLGEBRA DE ROBBINS

Los demostradores de teoremas se pueden aplicar a los problemas relacionados con la **verificación** y **síntesis** del *hardware* y *software*, porque ambos dominios pueden disponer de axiomatizaciones correctas. De este modo, la investigación en la demostración de teoremas se lleva a cabo en los campos del diseño del *hardware*, los lenguajes de programación, y la ingeniería del *software*, no sólo en la IA. En el caso del *software*, los axiomas definen las propiedades de cada elemento sintáctico del lenguaje de programación. (Razonar acerca de los programas es bastante parecido a razonar acerca de las acciones en el cálculo de situaciones.) Un algoritmo se verifica demostrando que sus salidas respetan las especificaciones de todas las entradas. El algoritmo RSA de encriptación de clave pública y el algoritmo de emparejamiento de texto de Boyer-Moore han sido verificados de este modo (Boyer y Moore, 1984). En el caso del *hardware*, los axiomas describen las interacciones entre las señales y los elementos del circuito. (*Véase* Capítulo 8 para un ejemplo.) El diseño de un sumador de 16 bits ha sido verificado por el AURA (Wojcik, 1983). Los razonadores especialmente diseñados para la verificación

VERIFICACIÓN

SÍNTESIS

han sido capaces de verificar CPUs enteras, incluyendo sus propiedades temporales (Srivas y Bickford, 1990).

La síntesis formal de algoritmos fue uno de los primeros usos de los demostradores de teoremas, tal como perfila Cordell Green (1969a), quien trabajó sobre las ideas iniciales de Simon (1963). La idea es demostrar un teorema de manera que «exista un programa *p* que satisfaga ciertas especificaciones». Si la demostración está restringida a ser constructiva el programa se puede obtener. Aunque, tal como se le llama, la **síntesis deductiva** no ha sido automatizada totalmente, y aún no ha conseguido ser viable para la programación de propósito general, la síntesis deductiva guiada ha tenido éxito en el diseño de diversos algoritmos bastante sofisticados y novedosos. Los programas de síntesis para propósito específico también son un área activa de investigación. En el área de síntesis del *hardware*, el demostrador de teoremas AURA se ha aplicado en el diseño de circuitos que son más compactos que cualquier diseño anterior (Wojciechowski y Wojcik, 1983). Para muchos diseños de circuitos, la lógica proposicional es suficiente porque el conjunto de proposiciones de interés está fijado por el conjunto de elementos del circuito. La aplicación de la inferencia proposicional en la síntesis del *hardware* es actualmente una técnica estándar que tiene muchas utilidades a gran escala (*véase*, por ejemplo, Nowick *et al*., (1993)).

SÍNTESIS DEDUCTIVA

Actualmente, estas mismas técnicas se están comenzando a aplicar en la verificación del *software*, por sistemas como el comprobador de modelos SPIN (Holzmann, 1997). Por ejemplo, el programa de control espacial del Agente Remoto fue verificado antes y después del vuelo (Havelund *et al*., 2000).

9.6 Resumen

Hemos presentado un análisis de la inferencia lógica en lógica de primer orden, y un número de algoritmos para realizarla.

- Un primer enfoque utiliza reglas de inferencia para instanciar los cuantificadores y proposicionalizar el problema de inferencia. Por lo general este enfoque es muy lento.
- El uso de la **unificación** para obtener las sustituciones adecuadas de las variables elimina el paso de instanciación en las demostraciones de primer orden, haciendo que el proceso sea mucho más eficiente.
- Una versión elevada del **Modus Ponens** utiliza la unificación para proporcionar una regla de inferencia natural y potente, el **Modus Ponens Generalizado**. Los algoritmos de **encadenamiento hacia delante** y de **encadenamiento hacia atrás** aplican esta regla a conjuntos de cláusulas positivas.
- El Modus Ponens Generalizado es completo para las cláusulas positivas, aunque el problema de la implicación es **semidecidible**. Para los programas **Datalog** que tienen cláusuas positivas con funciones libres, la implicación es decidible.
- El encadenamiento hacia delante se utiliza en las **bases de datos deductivas**, donde se puede combinar con las operaciones de las bases de datos relacionales. También se utiliza en los **sistemas de producción**, que pueden hacer actualizaciones eficientes en conjuntos de reglas muy grandes.

- El encadenamiento hacia delante es completo en los programas Datalog y corre en tiempo polinómico.
- El encadenamiento hacia atrás se utiliza en los **sistemas de programación lógica** como el **Prolog**, que emplea una sofisticada tecnología de compilación para proporcionar una inferencia muy rápida.
- El encadenamiento hacia atrás sufre de inferencias redundantes y bucles infinitos; esto se puede aliviar mediante la **memorización**.
- La regla de inferencia de la **resolución** generalizada proporciona un sistema de demostración completo en lógica de primer orden, utilizando bases de conocimiento en forma normal conjuntiva.
- Existen diversas estrategias para reducir el espacio de búsqueda de un sistema de resolución sin comprometer la completitud. Los demostradores de teoremas eficientes, basados en la resolución, se han utilizado para proporcionar teoremas matemáticos de interés y para verificar y diseñar *hardware* y *software*.

Notas bibliográficas e históricas

SILOGISMO

La inferencia lógica fue estudiada extensivamente por los matemáticos griegos. El tipo de inferencia estudiado más cuidadosamente por Aristóteles fue el **silogismo**, que es un tipo de regla de inferencia. Los silogismos de Aristóteles incluían elementos de la lógica de primer orden, como la cuantificación, pero estaba restringida a los predicados unarios. Los silogismos estaban categorizados mediante las «figuras» y los «modos», dependiendo del orden de los términos (a los que deberíamos llamar predicados) en las sentencias, el grado de generalidad (a lo que hoy en día deberíamos interpretar a través de los cuantificadores) aplicado a cada término, y si cada término estaba negado. El silogismo más importante es el primer modo de la primera figura:

Todo S es M.
Todo M es P.
Por lo tanto, todo S es P.

Aristóteles intentó demostrar la validez de otros silogismos mediante su «reducción» a aquellos de la primera figura. Era mucho menos preciso en describir lo que esta «reducción» debe implicar que en la caracterización de las propias figuras y modos silogísticos.

Gottlob Frege, quien desarrolló totalmente la lógica de primer orden en 1879, basó su sistema de inferencia en una gran colección de esquemas lógicamente válidos junto a una sola regla de inferencia, el Modus Ponens. Frege tomó ventaja del hecho de que el efecto de una regla de inferencia de la forma «De P, inferir Q» se puede simular aplicando el Modus Ponens a P junto a un esquema válido lógicamente como $P \Rightarrow Q$. Este estilo «axiomático» de exposición, utilizando el Modus Ponens junto a un número de esquemas lógicamente válidos, fue empleado por un número de lógicos después de Frege; y fue utilizado, de la forma más notable, en el *Principia Mathematica* (Whitehead y Russell, 1910).

Las reglas de inferencia, a diferencia de los esquemas axiomáticos, fueron el foco principal de la **deducción natural**, introducida por Gerhard Gentzen (1934) y por Sta-

nislaw Jaskowski (1934). A la deducción natural se le denomina «natural» porque no necesita la conversión a la (ilegible) forma normal, y porque sus reglas de inferencia intentan parecer cercanas a las personas. Prawitz (1965) proporciona un tratamiento extenso acerca de la deducción natural. Gallier (1986) utiliza el enfoque de Gentzen para exponer los cimientos teóricos de la deducción automática.

La invención de la forma clausular fue un paso crucial en el desarrollo profundo del análisis matemático en lógica de primer orden. Whitehead y Russell (1910) expusieron las denominadas *reglas de paso* (el término realmente pertenece Herbrand (1930)) que se utilizan para mover los cuantificadores al frente de las fórmulas. Las constantes de Skolem y las funciones de Skolem fueron introducidas lo suficientemente por Thoralf Skolem (1920). El procedimiento general de skolemización se dio por Skolem (1928), junto al importante concepto de universo de Herbrand.

El teorema de Herbrand, que recibe el nombre del lógico francés Jacques Herbrand (1930), ha jugado un papel vital en el desarrollo de los métodos de razonamiento automático, tanto antes como después de la introducción de la resolución de Robinson. Esto se refleja en nuestra referencia al «universo de Herbrand» más que al «universo de Skolem», aun pensando que fue realmente Skolem el que inventó el concepto. Herbrand también puede ser considerado como el inventor de la unificación. Gödel (1930) trabajó con base en las ideas de Skolem y Herbrand para demostrar que la lógica de primer orden tiene un procedimiento de demostración completo. Alan Turing (1936) y Alonzo Church (1936) demostraron, al mismo tiempo y utilizando diferentes demostraciones, que la validez en lógica de primer orden no era decidible. El excelente texto de Enderton (1972) explica todos estos resultados de un modo riguroso aunque de manera bastante comprensible.

Aunque McCarthy (1958) había sugerido el uso de la lógica de primer orden para la representación y el razonamiento en la IA, el primero de este tipo de sistemas fue desarrollado por los lógicos interesados en la demostración de teoremas matemáticos. Fue Abraham Robinson quien propuso el uso de la proposicionalización y del teorema de Herbrand, y Gilmore (1960) fue quien escribió el primer programa basado en este enfoque. Davis y Putnam (1960) utilizaron la forma clausular y desarrollaron un programa que intentaba encontrar refutaciones mediante la sustitución de las variables de los miembros del universo de Herbrand para generar cláusulas base y entonces averiguar si había inconsistencias entre las cláusulas base. Prawitz (1960) desarrolló la idea clave para permitir que el análisis de la inconsistencia proposicional dirija el proceso de búsqueda, y generar términos del universo de Herbrand sólo cuando es necesario hacerlo y así poder establecer la inconsistencia proposicional. Después de un largo desarrollo hecho por otros investigadores, esta idea le llevó a J. A. Robinson (no hay relación) a desarrollar el método de la resolución (Robinson, 1965). El denominado método inverso desarrollado por el investigador soviético S. Maslov (1964, 1967), más o menos en la misma época, y basado en unos principios algo diferentes, ofrece ventajas computacionales similares sobre la proposicionalización. El **método de conexión** de Wolfgang Bibel (1981) se puede ver como una extensión de este enfoque.

Después del desarrollo de la resolución, el trabajo sobre la inferencia en lógica de primer orden se produjo en varias direcciones diferentes. En la IA, la resolución se adoptó en los sistemas de respuesta a peticiones por Cordell Green y Bertram Raphael (1968).

Un enfoque algo menos formal se tomó por Carl Hewitt (1969). Su lenguaje PLANNER, aunque nunca implementado en su totalidad, fue un precursor de la programación lógica e incluía directivas para el encadenamiento hacia delante y hacia atrás, y para la negación como fallo. Un subconjunto conocido como MICRO-PLANNER (Sussman y Winograd, 1970) se implementó y utilizó en el sistema de comprensión del lenguaje natural SHRDLU (Winograd, 1972). Las implementaciones más recientes en IA ponen una buena cantidad de esfuerzo en las estructuras de datos que permitirían una recuperación eficiente de los hechos; este trabajo está cubierto en los textos de programación en IA (Charniak *et al*., 1987; Norvig, 1992; Forbus y de Kleer, 1993).

A principios de los 70, el **encadenamiento hacia delante** estaba bien establecido en la IA como una alternativa comprensible a la resolución. Se utilizó en una amplia variedad de sistemas, desde el demostrador de teoremas de geometría de Nevins (Nevins, 1975) al sistema experto R1 para la configuración de Vax (McDermott, 1982). Las aplicaciones en IA por lo general manejaban grandes cantidades de reglas, así que era importante desarrollar una tecnología eficiente para el emparejamiento de reglas, en concreto para las actualizaciones incrementales. La tecnología para los **sistemas de producción** se desarrolló para apoyar este tipo de aplicaciones. El lenguaje de sistema de producción OPS-5 (Forgy, 1981; Brownston *et al*., 1985) se utilizó para el R1 y para la arquitectura cognitiva SOAR (Laird *et al*., 1987). El OPS-5 incorporaba el proceso de emparejamiento rete (Forgy, 1982). El SOAR, que genera reglas nuevas para depositar los resultados de cálculos previos, puede crear conjuntos de reglas muy grandes, acerca de 1.000.000 de reglas en el caso del sistema TACAIR-SOAR para controlar un avión de caza simulado (Jones *et al*., 1998). CLIPS (Wygant, 1989) fue un lenguaje basado en C para sistemas de producción desarrollado en la NASA que permitió una mejor integración con otros paquetes de *software*, *hardware*, y sistemas de sensores, y se utilizó en la automatización de naves espaciales y diversas aplicaciones militares.

El área de investigación conocida como **bases de datos deductivas** también ha contribuido bastante a nuestro entendimiento de la inferencia hacia delante. Comenzó en una sesión de trabajo en Toulouse en 1977, organizada por Jack Minker, en la que juntó a expertos en inferencia lógica y en sistemas de bases de datos (Gallaire y Minker, 1978). Una revisión histórica reciente (Ramakrishnan y Ullman, 1995) dice, «los sistemas de [bases de datos] deductivas son un intento de adaptar el Prolog, que tiene un punto de vista del mundo de "datos pequeños", a un mundo de "datos grandes".» De este modo, pretende fundir la tecnología de bases de datos relacionales, que está diseñada para recuperar grandes *conjuntos* de hechos, con la tecnología de inferencia del Prolog, que por lo general recupera un hecho cada vez. Algunos de los textos sobre bases de datos deductivas son el de Ullman (1989) y el de Ceri *et al*., (1990).

Los trabajos influyentes de Chandra y Harel (1980) y el de Ullman (1985) condujeron a la adopción del Datalog como un lenguaje estándar para las bases de datos deductivas. La inferencia «de abajo arriba», o el encadenamiento hacia delante, también se convirtieron en el estándar, en parte porque evitan los problemas con la no terminación y el cálculo redundante que se presentan en el encadenamiento hacia atrás, y en parte porque tiene una implementación mucho más natural en términos de las operaciones de las bases de datos relacionales. El desarrollo de la técnica de los **conjuntos mágicos**

para la reescritura de las reglas de Bancilhon *et al*., (1986) le permitió al encadenamiento hacia delante coger prestado la ventaja de la orientación al objetivo del encadenamiento hacia atrás. Igualándolo a la carrera armamentística, los métodos de programación lógica tableados (*véase* Apartado 9.6) permitieron la ventaja de la programación lógica desde el encadenamiento hacia delante.

Gran parte de nuestro entendimiento de la complejidad de la inferencia lógica nos ha venido de la comunidad de las bases de datos deductivas. Chandra y Merlin (1977) fueron los primeros en demostrar que el emparejamiento de una única regla no recursiva (una **petición conjuntiva** en la terminología de las bases de datos) puede ser NP-duro. Kuper y Vardi (1993) propusieron la **complejidad de datos** (es decir, la complejidad como una función del tamaño de la base de datos, tratando el tamaño de la regla como constante) como una medida útil en la respuesta a una petición. Gottlob *et al*., (1999b) discuten la conexión entre las peticiones conjuntivas y la satisfacción de restricciones, mostrando cómo la descomposición de hiperárboles puede optimizar el proceso de emparejamiento.

PETICIÓN CONJUNTIVA

COMPLEJIDAD DE DATOS

Tal como hemos comentado anteriormente, el **encadenamiento hacia atrás** en la inferencia lógica apareció en el lenguaje PLANNER de Hewitt (1969). La programación lógica *per se* se desarrolló independientemente a este trabajo. Una forma restringida de la resolución lineal, denominada **resolución-SL** fue desarrollada por Kowalski y Kuehner (1971), construida a partir de la técnica de **eliminación de modelos** de Loveland (1968); cuando se aplicó a las cláusulas positivas, pasó a ser la **resolución-SLD**, que se presta a la interpretación de las cláusulas positivas como programas (Kowalski, 1974, 1979a, 1979b). Mientras tanto, en 1972, el investigador francés Alain Colmerauer había desarrollado e implementado **Prolog** con el propósito de analizar el lenguaje natural; las cláusulas de Prolog se pensaron inicialmente como reglas de gramáticas libres de contexto (Roussel, 1975; Colmerauer *et al*., 1973). Gran parte de la base teórica de la programación lógica se desarrolló por Kowalski, trabajando con Colmerauer. La definición semántica mediante mínimos puntos fijos es de Van Emden y Kowalski (1976). Kowalski (1988) y Cohen (1988) nos proporcionan unos buenos análisis históricos de los orígenes del Prolog. *Foundations on Logic Programming* (Lloyd, 1987) es un análisis teórico de los fundamentos del Prolog y otros lenguajes de programación lógica.

RESOLUCIÓN-SL

RESOLUCIÓN-SLD

Los compiladores eficientes para Prolog están basados, por lo general, en el modelo de cálculo de la *Warren Abstract Machine* (WAM) de David H. D. Warren (1983). Van Roy (1990) que mostró la aplicación de técnicas de compilación adicionales, como el tipo de inferencia, hizo programas en Prolog que eran competitivos con programas en C en lo referente a la velocidad. El proyecto *Japanese Fifth Generation*, un esfuerzo de investigación de 10 años, que comenzó en 1982, estaba totalmente basado en el Prolog para desarrollar sistemas inteligentes.

Los métodos para evitar los bucles innecesarios en los programas lógicos recursivos fueron desarrollados independientemente por Smith *et al*., (1986) y por Tamaki y Sato (1986). El último trabajo también incluía la memorización en los programas lógicos, un método desarrollado extensivamente como la **programación lógica tableada** por David S. Warren. Swift y Warren (1994) muestran cómo ampliar la WAM para manejar el tableado, permitiendo a los programas en Datalog ejecutarse en un orden de magnitud

más rápida que los sistemas de bases de datos deductivas mediante el encadenamiento hacia delante.

El trabajo teórico inicial sobre la programación lógica con restricciones fue de Jaffar y Lassez (1987). Jaffar *et al*., (1992a) desarrollaron el sistema CLP(R) para manejar restricciones con valores reales. Jaffar *et al*., (1992b) generalizaron la WAM para generar la CLAM (*Constraint Logic Abstract Machine*) para especificar las implementaciones del CLP. Ait-Kaci y Podelski (1993) describen un lenguaje sofisticado denominado LIFE, que combina CLP con la programación funcional y con el razonamiento sobre herencia. Kohn (1991) describe un proyecto ambicioso para utilizar la programación lógica con restricciones como el fundamento de una arquitectura de control en tiempo real, con aplicaciones de pilotos totalmente automatizados.

Hay un buen número de libros acerca de la programación lógica y el Prolog. *Logic for Problem Solving* (Kowalski, 1979b) es uno de los textos iniciales sobre la programación lógica en general. Entre los textos sobre el Prolog encontramos el de Clocksin y Mellish (1994), Shoham (1994) y Bratko (2001). Marriott y Stuckey (1998) proporcionan una excelente cobertura de la PLR. Hasta su cierre en 2000, la *Journal of Logic Programming* era la revista de partida; ahora ha sido reemplazada por *Theory and Practice of Logic Programming*. Entre las conferencias acerca de la programación lógica encontramos la *International Conference on Logic Programming* (ICLP) y el *International Logic Programming Symposium* (ILPS).

La investigación en **demostración de teoremas matemáticos** comenzó aún antes de los primeros sistemas de primer orden completos. El demostrador de teoremas de geometría de Herbert Gelernter (Gelernter, 1959) utilizaba los métodos de búsqueda heurística combinados con los diagramas de poda de subobjetivos falsos y era capaz de demostrar algunos resultados bastante intrincados de la geometría Euclidiana. Sin embargo, desde entonces, no ha habido mucha interacción entre la demostración de teoremas y la IA.

Los trabajos inciales se concentraban en la completitud. Siguiendo al trabajo inicial de Robinson, las reglas de demodulación y paramodulación para el razonamiento con igualdades fueron introducidas por Wos *et al*., (1967) y Wos y Robinson (1968), respectivamente. Estas reglas también fueron desarrolladas independientemente en el contexto de los sistemas de reescritura de términos (Knuth y Bendix, 1970). La incorporación del razonamiento con igualdad en el algoritmo de resolución se debe a Gordon Plotkin (1972); también era una característica del QLISP (Sacerdoti *et al*., 1976). Jouannaud y Kirchner (1991) hacen un repaso a la unificación ecuacional desde la perspectiva de la reescritura de términos. Los algoritmos eficientes de la unificación estándar fueron desarrollados por Martelli y Montanari (1976) y Paterson y Wegman (1978).

Junto al razonamiento con igualdad, los demostradores de teoremas han incorporado una variedad de procedimientos de propósito específico. Nelson y Oppen (1979) propusieron un esquema muy influyente para la integración de dichos procedimientos en un sistema de razonamiento general, entre otros métodos tenemos la «resolución de teoría» de Stickel (1985) y las «relaciones especiales» de Manna y Waldinger (1986).

Se han propuesto un buen número de estrategias para la resolución, comenzando con la estrategia de preferencia unitaria (Wos *et al*., 1964). La estrategia del conjunto soporte

fue propuesta por Wos *et al*., (1965), para proporcionar cierto grado de orientación al objetivo en la resolución. La resolución lineal apareció por primera vez en Loveland (1970). Genesereth y Nilsson (1987, Capítulo 5) proporcionan un breve, aunque cuidadoso, análisis de una gran variedad de estrategias de control.

Guard *et al*., (1969) describen el demostrador de teoremas SAM inicial, que ayudó a resolver un problema abierto acerca de la teoría de retículos. Wos y Winker (1983) dan un repaso a las contribuciones del demostrador de teoremas AURA a través de la resolución de problemas abiertos en diversas áreas de las matemáticas y la lógica. McCune (1992) continúa en esta dirección, hablando de las realizaciones del sucesor del AURA, el OTTER, en la resolución de problemas abiertos. Weidenbach (2001) describe el SPASS, uno de los demostradores de teoremas actuales más potentes. *A Computational Logic* (Boyer y Moore, 1979) es la referencia básica del demostrador de teoremas de Boyer-Moore. Stickel (1988) cubre la *Prolog Technology Theorem Prover* (PTTP), que combina las ventajas de la compilación del Prolog con la completitud de la eliminación de modelos (Loveland, 1968). SETHEO (Letz *et al*., 1992) es otro demostrador de teoremas ampliamente utilizado basado en este enfoque; puede realizar varios millones de inferencias por segundo en estaciones del modelo-2000. LEANTAP (Beckert y Posegga, 1995) es un demostrador de teoremas eficiente implementado tan sólo en 25 líneas de Prolog.

El trabajo inicial sobre síntesis automática de programas fue realizado por Simon (1963), Green (1969a) y Manna y Waldinger (1971). El sistema transformacional de Burstall y Darlington (1977) utilizaba el razonamiento ecuacional para la síntesis de programas recursivos. KIDS (Smith, 1990, 1996) es uno de los sistemas modernos más potentes; opera como un asistente experto. Manna y Waldinger (1992) dan una introducción de tutorial al estado del arte actual, con énfasis es su propio enfoque deductivo. *Automating Software Design* (Lowry y McCartney, 1991) recopila un buen número de trabajos en el área. El uso de la lógica en el diseño del *hardware* se trata en Kern y Greenstreet (1999); Clarke *et al*., (1999) cubre la comprobación de modelos para la verificación del *hardware*.

Computability and Logic (Boolos y Jeffrey, 1989) es una buena referencia sobre la completitud y la indecidibilidad. Muchos trabajos iniciales sobre lógica matemática se encuentran en *From Frege to Gödel: A Source Book in Mathematical Logic* (van Heijenoort, 1967). La revista de partida en este campo de la lógica matemática pura (opuesta a la deducción automática) es *The Journal of Symbolic Logic*. Entre los libros engranados en la deducción automática encontramos el clásico *Symbolic Logic and Mechanical Theorem Proving* (Chang y Lee, 1973), así como otros trabajos más recientes como el de Wos *et al*., (1992), Bibel (1993) y Kaufmann *et al*., (2000). La antología *Automation of Reasoning* (Siekmann y Wrightson, 1983) dispone de trabajos iniciales muy importantes sobre la deducción automática. Otras revisiones históricas han sido escritas por Loveland (1984) y Bundy (1999). La revista principal en este campo de la demostración de teoremas es la *Journal of Automated Reasoning*; la conferencia más importante es la *Conference on Automated Deduction* (CADE), que es anual. La investigación en la demostración de teoremas también está estrechamente relacionada con el uso de la lógica en el análisis de programas y los lenguajes de programación; y cuya conferencia más importante es la *Logic in Computer Science*.

EJERCICIOS

9.1 Demuestre a partir de los principios básicos que la Especificación Universal es sólida y que la Especificación Existencial genera una base de conocimiento inferencialmente equivalente.

9.2 De *Gusta*(*Jerry*, *Helado*) parece razonable inferir $\exists x$ *Gusta*(x, *Helado*). Apunte una regla de inferencia general, **Introducción del Existencial**, que sancione esta inferencia. Establezca cuidadosamente las condiciones que deben satisfacer las variables y los términos involucrados.

9.3 Suponga que una base de conocimiento contiene tan sólo una sentencia, $\exists x$ *TanAltoComo*(x, *Everest*). ¿Cuál de los siguientes resultados son legítimos al aplicar la Especificación Existencial?

a) *TanAltoComo*(*Everest*, *Everest*).
b) *TanAltoComo*(*Kilimanjaro*, *Everest*).
c) *TanAltoComo*(*Kilimanjaro*, *Everest*) $\wedge$ *TanAltoComo*(*BenNevis*, *Everest*) (después de dos aplicaciones).

9.4 Dé el unificador más general, si existe, de cada una de las siguientes parejas de sentencias atómicas:

a) $P(A, B, B)$, $P(x, y, z)$.
b) $Q(y, G(A, B))$, $Q(G(x, x), y)$.
c) *Mayor*(*Padre*(y), y), *Mayor*(*Padre*(x), *John*).
d) *Conoce*(*Padre*(y), y), *Conoce*(x, x).

9.5 Teniendo en cuenta los retículos de subsunción de la Figura 9.2.

a) Construya el retículo de la sentencia *Contrata*(*Madre*(*John*), *Padre*(*John*)).
b) Construya el retículo de la sentencia *Contrata*(*IBM*, y) («Cada uno trabaja en IBM»). Recuerde incluir cada tipo de petición que se unifica con la sentencia.
c) Asuma que ALMACENAR indexa cada sentencia bajo cada nodo de su retículo de subsunción. Explique cómo BUSCAR trabajaría cuando alguna de estas sentencias contiene variables; utilice como ejemplos las sentencias de (a) y la petición *Contrata*(x, *Padre*(x)).

9.6 Suponga que introducimos en una base de datos lógica un segmento de los datos del censo de Estados Unidos listando la edad, la ciudad de residencia, la fecha de nacimiento, y la madre de cada persona, utilizando los números de la seguridad social como las constantes de identificador de cada persona. De este modo, la edad de George sería *Edad*(443-65-1282, 56). ¿Cuál de los siguientes criterios de indexación C1-C5 permite una solución eficiente para cada una de las peticiones P1-P4 (asumiendo que se utiliza el encadenamiento hacia atrás estándar)?

- **C1**: un índice para cada átomo en cada posición.
- **C2**: un índice para cada primer argumento.
- **C3**: un índice para cada predicado atómico.
- **C4**: un índice para cada *combinación* de predicado y su primer argumento.

- **C5**: un índice para cada *combinación* de predicado y su segundo argumento y un índice para cada primer argumento (no estándar).

- **P1**: *Edad*(443-44-4321, x)
- **P2**: *Residen*(x, *Houston*)
- **P3**: *Madre*(x, y)
- **P4**: *Edad*(x, 34) $\wedge$ *Residen*(x, *TinyTownUSA*)

9.7 Uno podría pensar que se puede evitar el conflicto entre variables en la unificación durante el encadenamiento hacia atrás estandarizando todas las sentencias de la base de conocimiento de una vez por todas. Demuestre que para algunas sentencias este enfoque no puede aplicarse. (*Pista*: tenga en cuenta una sentencia, y una parte de ella que se unifique con otra.)

9.8 Explique cómo escribir cualquier problema de SAT-3 de tamaño arbitrario utilizando una única cláusula positiva de primer orden y no más de 30 hechos base.

9.9 Escriba representaciones lógicas para las siguientes sentencias, que sean adecuadas para utilizarse con el Modus Ponens Generalizado.

a) Los caballos, las vacas y los cerdos son mamíferos.
b) El descendiente de un caballo es un caballo.
c) Barba Azul es un caballo.
d) Barba Azul es padre de Charlie.
e) Descendiente y padre son relaciones inversas.
f) Cada mamífero tiene un padre.

9.10 En este ejercicio utilizaremos las sentencias que escribió en el Ejercicio 9.9 para responder a una pregunta mediante el algoritmo de encadenamiento hacia atrás.

a) Dibuje el árbol de demostración generado mediante un algoritmo de encadenamiento hacia atrás para la petición $\exists c$ *Caballo*(c), donde las cláusulas se emparejan en el orden dado.
b) ¿Qué nota en este dominio?
c) ¿Cuántas soluciones para c realmente se siguen de sus sentencias?
d) ¿Puede pensar en algún mecanismo para encontrarlas todas? (*Pista*: Podría querer consultar Smith *et al*., (1986).)

9.11 Un acertijo infantil muy popular dice «No tengo hermanos ni hermanas, pero el padre de ese hombre es el hijo de mi padre». Utilice las reglas del dominio de la familia (Capítulo 8) para demostrar quién es ese hombre. Puede aplicar cualquiera de los métodos descritos en este capítulo. ¿Por qué piensa que este acertijo es difícil?

9.12 Trace la ejecución del algoritmo de encadenamiento hacia atrás de la Figura 9.6 cuando se aplica para resolver el problema del crimen. Muestre la secuencia de los valores que se toman en la variable *objetivos*, y ordénelos en un árbol.

9.13 El siguiente código de Prolog define un predicado `P`:

```
P(X, [X|Y]).
P(X, [Y|Z] :- P(X, Z).
```

a) Muestre los árboles de demostración y las soluciones para las peticiones `P(A, [1, 2, 3])` y `P(2, [1, A, 3])`.

b) ¿Qué operación estándar sobre listas representa `P`?

9.14 En este ejercicio veremos la ordenación mediante el Prolog.

a) Escriba cláusulas en Prolog que definan el predicado `ordenado(L)`, que es verdadero si y sólo si la lista `L` está ordenada en orden ascendente.

b) Escriba una definición en Prolog del predicado `perm(L, M)`, que es verdadero si y sólo si `L` es una permutación de `M`.

c) Defina `ordena(L, M)` (`M` es la versión ordenada de `L`) utilizando `perm` y `ordenado`.

d) Ejecute `ordena` sobre listas más y más largas hasta que pierda la paciencia. ¿Cuál es la complejidad temporal de su programa?

e) Escriba un algoritmo de ordenación más rápido, como la ordenación por inserción o la ordenación rápida, en Prolog.

9.15 En este ejercicio veremos la aplicación recursiva de las reglas de reescritura mediante la programación lógica. Una regla de reescritura (o **demodulador** en la terminología del OTTER) es una ecuación con una dirección especificada. Por ejemplo, la regla de reescritura $x + 0 \rightarrow x$ sugiere reemplazar cualquier expresión que empareje con $x + 0$ por la expresión x. La aplicación de las reglas de reescritura es una parte central de los sistemas de razonamiento ecuacionales. Utilizaremos el predicado `rescribir(X; Y)` para representar las reglas de reescritura. Por ejemplo, la regla de reescritura anterior se escribe `rescribir(X + 0, X)`. Algunos términos son *primitivos* y por tanto no se pueden simplificar más; por tanto, escribiremos `primitivo(0)` para decir que 0 es un término primitivo.

a) Escriba una definición de un predicado `simplificar(X, Y)`, que es verdadero cuando `Y` es una versión simplificada de `X`, es decir, cuando no hay más reglas de reescritura aplicables a cualquier subexpresión de `Y`.

b) Escriba un conjunto de reglas para la simplificación de expresiones que tengan operadores aritméticos, y aplique su algoritmo de simplificación a algunas de las expresiones de muestra.

c) Escriba un conjunto de reglas de reescritura para la diferenciación simbólica, y utilícelas entre sus reglas de simplificación para diferenciar y simplificar expresiones que tengan expresiones aritméticas, incluyendo la exponenciación.

9.16 En este ejercicio vamos a tener en cuenta la implementación de los algoritmos de búsqueda en Prolog. Suponga que `sucesor(X, Y)` es verdadero cuando `Y` es el estado sucesor del estado `X`; y que `objetivo(X)` es verdadero cuando `X` es el estado objetivo. Escriba una definición para `resolver(X, C)`, que significa que `C` es un camino (lista de estados) comenzando por `X` y acabando en el estado objetivo, y que consiste en una secuencia de pasos legales tal como se definen mediante el predicado sucesor. Encontrará que la búsqueda del primero en profundidad es la forma más fácil de hacerlo. ¿Cómo de fácil sería añadir una heurística para el control de la búsqueda?

9.17 ¿Cómo se puede utilizar la resolución para demostrar que una sentencia es válida?, ¿e insatisfacible?

9.18 De «Los caballos son animales», se sigue que «La cabeza de un caballo es la cabeza de un animal». Demuestre que esta inferencia es válida llevando a cabo los siguientes pasos:

a) Traduzca la premisa y la conclusión al lenguaje de la lógica de primer orden. Utilice tres predicados: *CabezaDe*(c, x) (que significa que «c es la cabeza de x»), *Caballo*(x) y *Animal*(x).
b) Niegue la conclusión y transforme la premisa y la conclusión negada a la forma normal conjuntiva.
c) Utilice la resolución para demostrar que la conclusión se sigue de la premisa.

9.19 Aquí tenemos dos sentencias en el lenguaje de la lógica de primer orden:

$$\textbf{(A)}: \forall x\, \exists y\, (x \geqslant y)$$
$$\textbf{(B)}: \exists y\, \forall x\, (x \geqslant y)$$

a) Asuma que el rango de las variables está sobre todos los números naturales 0, 1, 2, …, ∞ y que el predicado «⩾» significa «es mayor o igual a». Bajo esta interpretación, transforme (A) y (B) al lenguaje natural.
b) ¿Es (A) verdadero bajo esta interpretación?
c) ¿Es (B) verdadero bajo esta interpretación?
d) ¿(A) implica lógicamente (B)?
e) ¿(B) implica lógicamente (A)?
f) Utilizando la resolución, intente demostrar que (A) se sigue de (B). Hágalo aunque piense que (B) no implica lógicamente (A); continúe hasta que la demostración se interrumpa y no pueda proceder (si se ha interrumpido). Muestre que unificación y sustitución en cada paso de la resolución. Si la demostración falla, explique concretamente dónde, cómo y por qué ha fallado.
g) Ahora intente demostrar que (B) se sigue de (A).

9.20 La resolución puede generar demostraciones no constructivas a partir de peticiones con variables, así que teníamos que introducir mecanismos especiales para extraer las respuestas positivas. Explique por qué este problema no aparece en las bases de datos que sólo contienen cláusulas positivas.

9.21 En este capítulo hemos comentado que la resolución no se puede utilizar para generar todas las consecuencias lógicas de un conjunto de sentencias. ¿Algún algoritmo lo puede hacer?

10 Representación del conocimiento

Donde se muestra cómo utilizar la lógica de primer orden para representar los aspectos más importantes del mundo real como las acciones, el espacio, el tiempo, los eventos mentales y el hecho de ir de compras.

En los últimos tres capítulos se han descrito las tecnologías en las que se sustentan los agentes basados en el conocimiento: la sintaxis, la semántica y las demostraciones teóricas de la lógica proposicional y de primer orden, así como la implementación de los agentes que utilizan este tipo de lógica. En este capítulo se abordará la cuestión de qué *contenido* incorporar a la base de conocimiento de los agentes (cómo representar hechos acerca del mundo).

La Sección 10.1 introduce la idea general de ontología, que organiza todo lo existente en el mundo en una jerarquía de categorías. La Sección 10.3 aborda la representación de las acciones, aspecto fundamental en la construcción de agentes basados en conocimiento. La Sección 10.2 cubre las categorías básicas de objetos y sustancias, y la Sección 10.3 explica el concepto más general de **eventos**, o segmentos espacio-temporales. La Sección 10.4 habla sobre el conocimiento acerca de las creencias y la Sección 10.5 proporciona todo el conocimiento de forma conjunta en el contexto de una tienda que vende a través de Internet. Los Apartados 10.6 y 10.7 cubren los sistemas de razonamiento especializados para representar incerteza y conocimiento cambiante.

10.1 Ingeniería ontológica

En dominios de «juguete», el problema de la representación no es importante y es fácil encontrar un vocabulario consistente. Por otro lado, los dominios complejos como son

la compra utilizando Internet o el control de un robot en un entorno físico cambiante, requieren representaciones más generales y flexibles. Este capítulo muestra cómo crear estas representaciones, concentrándose en conceptos generales (como las *acciones*, el *tiempo*, los *objetos físicos* y las *creencias*) que ocurren en dominios muy diferentes. La representación de estos conceptos abstractos se suele denominar **ingeniería ontológica** (relacionada con el proceso de la **ingeniería del conocimiento** descrito en la Sección 8.4, aunque opera a gran escala).

INGENIERÍA ONTOLÓGICA

La posibilidad de representarlo *todo* en el mundo, es una tarea de enormes proporciones. Por supuesto, no se va a realizar una descripción completa de todo (eso sería demasiado hasta para un libro de 1.000 páginas), pero se dejarán moldes donde se pueda incorporar nuevo conocimiento, sea cual sea el dominio. Por ejemplo, se definirá lo que significa un objeto físico, y los detalles de diferentes tipos de objetos (robots, televisores o cualquier otra cosa), que pueda ser rellenado con posterioridad. El marco de trabajo general para los conceptos se llama **ontología superior**, debido a la convención general de representar en los grafos los conceptos generales en la parte superior y los conceptos más específicos debajo de ellos, como en la Figura 10.1.

ONTOLOGÍA SUPERIOR

Antes de considerar la ontología en más profundidad, se planteará una advertencia importante. Se ha seleccionado el uso de la lógica de primer orden para tratar el contenido y la organización del conocimiento. Ciertos aspectos del mundo real son difíciles de capturar en LPO (lógica de primer orden). La principal dificultad es que casi todas las generalizaciones tienen excepciones, o son ciertas sólo en un determinado grado. Por ejemplo, aunque «los tomates son rojos» es una regla útil, algunos tomates son verdes, amarillos o naranjas. Se pueden encontrar excepciones similares para casi todas las afirmaciones hechas en este capítulo. La habilidad para manejar excepciones e incerteza es extremadamente importante, pero es ortogonal con la tarea de comprender la ontología general. Por esta razón, se retrasará el tratamiento de excepciones

Figura 10.1 La ontología superior del mundo, en donde se indican los temas que más adelante se abordan en el capítulo. Cada arco indica que el concepto inferior es una especialización del concepto superior.

hasta la Sección 10.6, y los aspectos más generales sobre información incierta hasta el Capítulo 13.

¿Qué se utiliza en una ontología superior? Considérese de nuevo la ontología para circuitos de la Sección 8.4. Se realiza un gran número de asunciones para su simplificación. Por ejemplo, el tiempo se omite por completo. Las señales son fijas y no se propagan. La estructura de un circuito se mantiene constante. Si se quiere hacer más general, se considerarán las señales a distintos intervalos de tiempo y se incluirá la longitud de los cables y los retrasos de propagación. Esto permitiría simular las propiedades del tiempo en el circuito, y de hecho esas simulaciones son a menudo llevadas a cabo por diseñadores de circuitos. También se podrían introducir clases de puertas más interesantes, por ejemplo especificando la tecnología (TTL, MOS, CMOS, etc.), así como las especificaciones de entrada/salida. Si se quisiera considerar la fiabilidad del diagnóstico, se introduciría la posibilidad de que la estructura del circuito o las propiedades de las puertas pudieran cambiar de forma espontánea. Para tener en cuenta las capacitaciones perdidas, se debería ir desde una representación puramente topológica hacia una descripción más realista de las propiedades geométricas.

Al centrarse en el mundo de los *wumpus*, se aplican consideraciones similares. Aunque se incluye el tiempo, tiene una estructura muy simple: nada sucede excepto cuando el agente actúa y todos los cambios son instantáneos. Una ontología más general, que se adapte mejor para el mundo real, debería permitir cambios simultáneos que se extienden en el tiempo. Se utiliza el predicado *Hoyo* para especificar qué cuadrados tienen hoyos. Se podrían haber permitido diferentes clases de hoyos teniendo varios individuos que pertenecieran a dichas clases, cada uno de ellos con diferentes propiedades. De igual modo, se podría querer permitir otros animales además de *wumpuses*. No sería posible identificar las especies exactas a partir de las percepciones, por lo tanto sería necesario construir una taxonomía biológica del mundo de los *wumpus* para ayudar al agente a predecir el comportamiento partiendo de pistas escasas.

Para una ontología de propósito específico, es posible hacer cambios como estos para moverse hacia una mayor generalidad. Entonces surge una pregunta obvia: ¿convergerán todas estas ontologías en una ontología de propósito general? Después de siglos de investigación filosófica y computacional, la respuesta es «posiblemente». En esta sección, se propondrá una versión, que representa una síntesis de las ideas de todos estos siglos. Hay dos características principales en las ontologías de propósito general que las distinguen de la colección de ontologías de propósito específico:

- Una ontología de propósito general debe ser aplicable en mayor o menor medida a cualquier dominio de propósito específico (con la inclusión de axiomas específicos del dominio). Esto significa que, en tanto como sea posible, no se deben refinar aspectos de representación ni ser ignorados.
- En un dominio dispar, las diferentes áreas de conocimiento deben ser *unificadas*, puesto que el razonamiento y la resolución de problemas podría involucrar varias áreas simultáneamente. Un sistema de reparación de circuitos para robots, por ejemplo, necesitar razonar acerca de los circuitos en términos de conectividad eléctrica y disposición física, y sobre el tiempo para realizar el análisis de tiempos y estimar el costo de la obra. Las sentencias que describen el tiempo deben ser ca-

paces de poder ser combinadas con aquellas que describen disposición espacial, y deben trabajar igualmente bien con nanosegunos y minutos o utilizando angstroms y metros.

Después de presentar la ontología general, se utilizará para describir el dominio de compra por Internet. Este dominio es más que adecuado para ejercitar la ontología propuesta, y proporciona el alcance suficiente para que el lector realice alguna representación creativa de conocimiento por su cuenta. Considérese por ejemplo, que el agente de compra por Internet debe conocer millares de títulos y autores para comprar libros en Amazon.com, todas las clases de comidas para comprar provisiones en Peapod.com y todo lo que cualquiera puede encontrarse en una estación de servicio para buscar gangas en Ebay.com[1].

10.2 Categoría y objetos

CATEGORÍAS

La organización de objetos en **categorías** es una parte vital de la representación del conocimiento. Aunque la interacción con el mundo tiene lugar a nivel de objetos individuales, *la mayoría del proceso de razonamiento tiene lugar en el nivel de categorías*. Por ejemplo, un comprador puede tener el objetivo de comprar un balón de baloncesto, en lugar de un balón de baloncesto *concreto* como BB_9. Las categorías también sirven para hacer predicciones sobre los objetos una vez que están clasificados. Se puede inferir la presencia de ciertos objetos a través de la percepción, inferir la categoría a la que pertenece utilizando las propiedades del objeto percibidas y entonces usar las información sobre categorías para realizar predicciones sobre los objetos. Por ejemplo, a partir de las características verde, cáscara moteada, tamaño grande y con forma ovalada, uno puede inferir que un objeto es una sandía; a partir de esto, uno puede inferir que puede ser útil para una ensalada de frutas.

Existen dos opciones para representar categorías en lógica de primer orden: predicados y objetos. Es decir, se puede usar el predicado *Balón_de_baloncesto*(*b*), o se puede **reformular** la categoría como un objeto, *Balones_de_baloncesto*. Entonces se puede decir *Miembro(b, Balones_de_Baloncesto)* (que se puede abreviar como $b \in$ *Balones_de_baloncesto*) para decir que *b* es un miembro de la categoría de balones de baloncesto. Se utiliza *Subconjunto(Balones_de_baloncesto, Balones)* (abreviado como *Balones_de_baloncesto* $\subset$ *Balones*) para indicar que los balones de baloncesto son una subcategoría, o subconjunto, de los *Balones*. Entonces se puede pensar que una categoría es un conjunto que agrupa a sus miembros, o se puede pensar que es un objeto más complejo que surge cuando tienen sentido las relaciones de *Miembro* y *Subconjunto* definidas para él.

HERENCIA

Las categorías sirven para organizar y simplificar el conocimiento base, a través de la **herencia**. Se dice que todos los objetos de la categoría *Alimentos* son comestibles, y

[1] Discúlpese si debido a circunstancias fuera de nuestro control, algunas de las tiendas *on-line* no se encuentran funcionando en el momento que el lector lea estas líneas.

se afirma que *Fruta* es una subclase de *Alimentos* y que *Manzanas* en una subclase de *Fruta*, entonces se sabe que cualquier manzana es comestible. Se dice que las manzanas individuales **heredan** la propiedad comestible, es este caso por su función de pertenencia a la categoría *Alimentos*.

TAXONOMÍA

Las relaciones de subclasificación organizan categorías en **taxonomías**, o **relaciones taxonómicas**. Las taxonomías se han utilizado explícitamente desde hace siglos en campos técnicos. Por ejemplo, la biología sistemática intenta proporcionar una taxonomía para todas las especies vivas y extinguidas; la bibliografía de ciencias ha desarrollado una taxonomía de todos los campos del conocimiento, codificado como sistema Decimal de Dewey, mientras que los departamentos de impuestos y otros departamentos gubernamentales han desarrollado grandes taxonomías sobre ocupaciones y productos comerciales. Las taxonomías son también un aspecto importante en el conocimiento general del sentido común.

La lógica de primer orden hace sencillo realizar afirmaciones sobre categorías, ya sea relacionando objetos con categorías o cuantificando sus miembros:

- Un objeto es un miembro de una categoría. Por ejemplo:
$$BB_9 \in Balones_de_baloncesto$$
- Una categoría es subclase de otra categoría. Por ejemplo:
$$Balones_de_baloncesto \subset Balones$$
- Todos los miembros de una categoría tienen algunas propiedades. Por ejemplo:
$$x \in Balones_de_baloncesto \Rightarrow Redondo(x)$$
- Miembros de una categoría se pueden reorganizar por algunas propiedades. Por ejemplo:
$$Naranja(x) \wedge Redondo(x) \wedge Diámetro(x) = 9.5'' \wedge x \in Balones \Rightarrow x \in$$
$$\in Balones_de_baloncesto$$
- Una categoría como conjunto tiene algunas propiedades. Por ejemplo:
$$Perros \in EspeciesDomesticadas$$

Nótese que debido a que *Perros* es una categoría y es un miembro de *EspeciesDomesticadas*, esta última debe ser una categoría de categorías. Se podrían tener incluso categorías de categorías de categorías, pero no son de mucha utilidad.

Aunque las relaciones de subclasificación y miembro son las más importantes para las categorías, también se quiere ser capaz de modelar relaciones entre categorías que no son subclase unas de otras. Por ejemplo, si se dice que *Machos* y *Hembras* son subclases de *Animales*, entonces no se afirma que *Machos* no sean *Hembras*. Se dice que dos o más categorías son **disjuntas**, si no tienen miembros en común. Incluso si se conoce que machos y hembras son disjuntos, no se sabe que un animal que no es un macho, debe ser una hembra, a menos que se explicite que machos y hembras constituyen una **descomposición exhaustiva** de los animales. Una descomposición exhaustiva disjunta se conoce como una **partición**. Los siguientes ejemplos ilustran estos tres conceptos:

DISJUNTAS

DESCOMPOSICIÓN EXHAUSTIVA

PARTICIÓN

$$Disjunto(\{Animales, Vegetales\})$$
$$DescomposiciónExhaustiva(\{Americanos, Canadienses, Mejicanos\},$$
$$Norteamericanos)$$
$$Partición(\{Machos, Hembras\}, Animales)$$

(Nótese que la *DescomposiciónExhaustiva* de *Norteamericanos* no es una *Partición*, porque alguna gente tiene doble nacionalidad.) Los tres predicados se definen a continuación:

$$Disjunto(s) \Leftrightarrow (\forall\ c_1, c_2\ \ c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow Intersección(c_1, c_2) = \{\ \})$$
$$DescomposiciónExhaustiva(s, c) \Leftrightarrow (\forall\ i\ \ i \in c \Leftrightarrow \exists\ c_2\ c_2 \in s \wedge i \in c_2)$$
$$Partición(s, c) \Leftrightarrow Disjunto(s) \wedge DescomposiciónExhaustiva(s, c)$$

Las categorías también se pueden definir proporcionando las condiciones necesarias y suficientes para la función de pertenencia. Por ejemplo, soltero es un macho adulto no casado:

$$x \in Solteros \Leftrightarrow NoCasado(x) \wedge x \in Adultos \wedge x \in Machos$$

Como se comenta en el recuadro correspondiente al género natural, no siempre es posible formular definiciones lógicas rigurosas de las categorías, ni en todos los casos es necesario.

Objetos compuestos

La idea de que un objeto puede ser parte de otro es familiar. La nariz forma parte de la cabeza, Rumanía es parte de Europa y este capítulo es parte de este libro. En general, se utiliza la relación *ParteDe* para decir que algo forma parte de otra cosa. Los objetos se pueden agrupar dentro de jerarquías *ParteDe*, reminiscencia de la jerarquía *Subconjunto*:

$$ParteDe(Bucarest, Rumanía)$$
$$ParteDe(Rumanía, EuropaDelEste)$$
$$ParteDe(EuropaDelEste, Europa)$$
$$ParteDe(Europa, Tierra)$$

La relación *ParteDe* es transitiva y reflexiva: es decir,

$$ParteDe(x, y) \wedge ParteDe(y, z) \Rightarrow ParteDe(x, z)$$
$$ParteDe(x, x)$$

Por lo tanto, se puede concluir *ParteDe*(*Bucarest*, *Tierra*).

OBJETOS COMPUESTOS

Las categorías de **objetos compuestos** se caracterizan a menudo por relaciones estructurales entre las partes. Por ejemplo, un bípedo tiene dos piernas unidas a su cuerpo:

$$\begin{aligned} Bipedo(a) \quad \Rightarrow \quad & \exists\ l_1, l_2, b\ Pierna(l_1) \wedge Pierna(l_2) \wedge Cuerpo(b) \wedge \\ & ParteDe(l_1, a) \wedge ParteDe(l_2, a) \wedge ParteDe(b, a) \wedge \\ & UnidaA(l_1, b) \wedge UnidaA(l_2, b) \wedge \\ & l_1 \neq l_2 \wedge [\forall\ l_3\ Pierna(l_3) \wedge ParteDe(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)] \end{aligned}$$

La notación de «exactamente dos» no es la más adecuada. Esto fuerza a decir que hay dos piernas, que no son las mismas, y que si alguien propone una tercera pierna, deberá ser la misma que una de las otras dos. En la Sección 10.6, se verá cómo un formalismo denominado descripción lógica hace mucho más fácil representar restricciones como «exactamente dos».

Se puede definir una *PparticiónDePartes* como análoga a la relación de *Partición* correspondiente a las categorías (*véase* Ejercicio 10.6). Los objetos están constituidos por las partes de su *PparticiónDePartes*, y se puede considerar que algunas de sus propiedades se derivan de tales partes. Por ejemplo, la masa de un objeto compuesto, es la suma de cada una de sus partes. Conviene advertir que no sucede lo mismo con las categorías, las cuales no tienen masa, aunque sus elementos sí puedan tenerla.

Es conveniente definir los objetos compuestos mediante partes bien definidas, aunque sin una estructura determinada. Por ejemplo, se podría afirmar: «Las manzanas de esta bolsa pesan kilo y medio». La tentación sería adscribir este peso al conjunto de manzanas en la bolsa, pero esto es un error, porque el conjunto es un concepto matemático abstracto que tiene elementos pero no tiene peso. En su lugar, se precisa de un nuevo concepto, el cual se llamará **montón**. Por ejemplo, si las manzanas son $Manzana_1$, $Manzana_2$ y $Manzana_3$, entonces

MONTÓN

$$MontónDe(\{Manzana_1, Manzana_2, Manzana_3\})$$

denota al objeto compuesto cuyas partes son las tres manzanas (no elementos). Se puede utilizar el concepto de montón como un objeto normal, aunque no estructurado. Nótese que *MontónDe*(*Manzanas*) es un objeto compuesto formado por todas las manzanas (que no debe ser confundido con *Manzanas*, la categoría o conjunto de todas las manzanas).

Se puede definir *MontónDe* en términos de la relación *ParteDe*. Obviamente, cada elemento de s es parte de $MontónDe(s)$:

$$\forall\, x \;\; x \in s \Rightarrow ParteDe(s, MontónDe(s))$$

Además, $MontónDe(s)$ es el objeto más pequeño que satisface esta condición. En otras palabras, $MontónDe(s)$ debe ser parte de cualquier objeto que tiene todos los elementos de s como partes:

$$\forall\, y\, [\forall\, x \;\; x \in s \Rightarrow ParteDe(x, y)] \Rightarrow ParteDe(MontónDe(s), y)$$

MIIMIZACIÓN LÓGICA

Estos axiomas son un ejemplo de una técnica general llamada **minimización lógica**, que define a un objeto como el más pequeño que satisface ciertas condiciones.

Medidas

Tanto en las teorías científicas del mundo como en las que apelan al sentido común, los objetos poseen peso, masa, costo, etc. Los valores que se asignan a estas propiedades se conocen como **medidas**. Es muy fácil representar medidas cuantitativas. Se puede pensar que en el universo existen «objetos de medida» abstractos tales como la *longitud*, que es la longitud de este segmento de línea: ├──────────────────────┤. A la longitud anterior se puede llamar 1,5 pulgadas o 3,81 centímetros. Es decir, la misma longitud puede denominarse de diferentes formas en el lenguaje. Lógicamente, esto se realiza combinando una **función de unidades** con un número (un esquema alternativo se explora en el Ejercicio 10.8). Si se denota al segmento de línea como L_1, se puede escribir

MEDIDAS

FUNCIÓN DE UNIDADES

$$Longitud(L_1) = Pulgadas(1{,}5) = Centímetros(3{,}81)$$

LOS GÉNEROS NATURALES

Algunas categorías se definen de manera rigurosa: un objeto se considera como triángulo si y sólo si es un polígono de tres lados. Por el contrario, la mayor parte de las categorías del mundo real que no tienen una definición precisa se llaman de **género natural.** Por ejemplo, los tomates en general tienen un color escarlata tenue, son más o menos esféricos y tienen una pequeña depresión en la parte superior, que es donde está el tallo, tienen un diámetro que puede ir desde 7.5 hasta 10 cm, tienen piel delgada pero resistente y en el interior tienen pulpa, semillas y jugo. Aunque también hay variaciones: algunos tomates son de color naranja, los tomates que no están todavía maduros son verdes, algunos son más grandes y otros más pequeños que el promedio, mientras que los tomates pequeños de ensalada son todos igual de pequeños. En vez de una definición completa de los tomates, se tiene un conjunto de características que permiten identificar esos objetos, que evidentemente son tomates típicos. Características que no permitirán tomar decisiones en el caso de otros objetos. ¿Acaso puede haber tomates con piel de melocotón?

Lo anterior plantea un problema al agente lógico. Éste no puede estar seguro de que aquello que ha percibido sea un tomate, incluso estando seguro de ello, no tendría la certeza de qué propiedades de un tomate típico tiene ese tomate en particular. El problema anterior es consecuencia inevitable de operar en entornos parcialmente observables.

La idea clave consiste en separar lo que es válido para todos los casos concretos de una categoría, de aquello que sólo se cumple en los casos típicos de esa categoría. Por ejemplo, además de la categoría *Tomates*, también se dispone de la categoría *Típicos*(*Tomates*). En este caso, *Típico* es una función que correlaciona una categoría con la subclase de tal categoría en la que se encuentran sólo los casos típicos:

$$Típico(c) \subseteq c$$

De hecho, gran parte del conocimiento sobre los géneros naturales se refiere a los casos típicos:

$$x \in Típico(Tomates) \Rightarrow Rojo(x) \wedge Redondo(x)$$

De esta forma, se pueden poner por escrito hechos útiles acerca de la categorías, sin tener que ofrecer definiciones exactas.

Wittgenstein (1953), en su libro *Philosophical Investigations*, explicó lo difícil que es ofrecer definiciones exactas para la mayoría de las categorías naturales. Empleó el ejemplo de los juegos, para mostrar que los miembros de una categoría lo que tenían en común eran «parecidos familiares», más que características necesarias y suficientes.

Quine (1953), desafió también la utilidad de la noción de definición rigurosa. Comentó que, incluso una definición como la anterior de «soltero», deja que desear. Por ejemplo, cuestionó afirmaciones como la de que «el Papa es soltero». Aunque esta afirmación no es estrictamente falsa, su uso es desafortunado porque induce a inferencias no deseadas por parte del receptor. El conflicto podría ser resuelto distinguiendo entre definiciones lógicas adecuadas para representación de conocimiento interno, y aquellos criterios más matizados para su correcto uso lingüístico. Éstos últimos podrían ser alcanzados filtrando las aserciones derivadas de las primeras. También puede ser posible que los fallos derivados del uso lingüístico sirvan como retroalimentación para modificar definiciones internas, filtrando aquellas que no sean necesarias.

La conversión entre una unidad y la otra se realiza igualando múltiplos de una unidad respecto a la otra:

$$Centímetros(2{,}54 \times d) = Pulgadas(d)$$

Se pueden escribir axiomas similares para libras y kilogramos, segundos y días, así como dólares y centavos. Las medidas se pueden usar para describir objetos, como por ejemplo:

$$Diámetro(Balón_de_baloncesto_{12}) = Pulgadas(9.5)$$
$$Precio(Balón_de_baloncesto_{12}) = \$(19)$$
$$d \in Días \Rightarrow Duración(d) = Horas(24)$$

Es conveniente resaltar que $(1) *no* es un billete de un dólar. Uno puede tener dos billetes de dólar, pero hay sólo un objeto denominado $(1). Resaltar también que mientras *Pulgadas*(0) y *Centímetros*(0) se refieren al mismo valor cero de longitud, no son equivalentes a otras medidas de cero como *Segundos*(0).

Es fácil representar las medidas sencillas y cuantitativas. Existe otro tipo de medidas que son más difíciles, pues no se dispone de una escala de valores bien definida. Los ejercicios tienen dificultad, los postres tienen delicia, los poemas tienen belleza, sin embargo a ninguna de estas cualidades se le puede asignar un número. Se podría estar tentado, en un afán por tener en cuenta sólo aquello que es contable, a descartar las propiedades anteriores al considerarlas inútiles para el razonamiento lógico, o lo que es peor, imponer una escala numérica a la belleza. Lo anterior sería un grave error, puesto que no es necesario. El aspecto más importante de las medidas no reside en los valores numéricos particulares en sí, sino en el hecho de que las medidas permiten una *ordenación*.

Aun cuando las medias no estén representadas por números, es posible compararlas entre sí mediante signos de ordenación como $>$. Por ejemplo, hay quienes consideran que los ejercicios de Norvig son más difíciles que los propuestos por Russell, y es más difícil obtener una buena calificación en éstos:

$$e_1 \in Ejercicios \wedge e_2 \in Ejercicios \wedge Escribió(Norvig, e_1) \wedge Escribió(Russell, e_2) \Rightarrow Dificultad(e_1) > Dificultad(e_2)$$
$$e_1 \in Ejercicios \wedge e_2 \in Ejercicios \wedge Dificultad(e_1) > Dificultad(e_2) \Rightarrow CalificaciónEsperada(e_1) < CalificaciónEsperada(e_2)$$

Lo anterior bastará para decidir qué ejercicios realizar, aunque no existan valores numéricos para tomar tal decisión (lo que sí sería necesario es saber quién escribió cada ejercicio). Este tipo de relaciones monotónicas que guardan entre sí las medidas, constituye la base del campo conocido como **física cualitativa**, un subcampo de la IA que investiga cómo razonar acerca de los sistemas físicos, sin tener que enfrascarse en la elaboración de minuciosas ecuaciones y simulaciones numéricas. En la sección de notas históricas se habla sobre la física cualitativa.

Sustancias y objetos

Quizás al mundo real se le podría considerar constituido por objetos primitivos (partículas) y por objetos compuestos construidos con estas partículas. Al razonar en el nivel

de objetos grandes como manzanas y coches, se elimina la complejidad de tratar por separado con una inmensa cantidad de objetos primitivos. Existe sin embargo, una importante porción de realidad que parecería desafiar todo tipo de **individualización**: separación en objetos distintos. A esta porción se le conoce genéricamente como **materia o sustancia**. Por ejemplo, supóngase que ante mí tengo un oso hormiguero y mantequillas. Si bien se puede afirmar que hay un oso hormiguero, no es obvio cómo se puede cuantificar la mantequilla, no es evidente qué cantidad de «objetos-mantequilla» hay, puesto que cualquier parte de un objeto-mantequilla, es también otro objeto-mantequilla, por lo menos hasta llegar a partes realmente diminutas. Lo anterior constituye la diferencia fundamental entre una sustancia y las cosas. Si se parte por la mitad al oso hormiguero, no resultan dos osos hormigueros, desafortunadamente.

INDIVIDUALIZACIÓN

MATERIA O SUSTANCIA

Nótese que en español, como en otros idiomas, se distingue claramente entre sustancias y cosas. Se dice, por ejemplo, «un oso hormiguero», en cambio (excepto en algunos presuntuosos restaurantes californianos) nunca se pediría «una mantequilla». Los lingüistas establecen una diferencia entre **sustantivos contables** como osos hormigueros, orificios y teoremas y **sustantivos no contables** como mantequilla, agua y energía. Varias ontologías competentes reclaman el manejo de esta distinción. Aquí se describirá sólo una, las restantes se tratan en la sección de notas históricas.

SUSTANTIVOS CONTABLES

SUSTANTIVOS NO CONTABLES

Para representar correctamente a las sustancias, se tiene que empezar por lo que es obvio. En la ontología presentada, se tendrán que incluir por lo menos el grueso del «paquete» de aquellas sustancias con las que se interactúa. Por ejemplo, se podría considerar que la mantequilla es la misma que la que se dejó sobre la mesa la noche anterior, se puede coger, pesarla, venderla o cualquier otra cosa. En este sentido, la mantequilla es un objeto igual que el oso hormiguero. Llámese $Mantequilla_3$. Se definirá también la categoría *Mantequilla*. De manera no formal, sus elementos serán todas aquellas cosas de las que se puede afirmar «Es mantequilla», incluida $Mantequilla_3$. Aclarando que por ahora se omitirán algunas partes muy pequeñas, toda parte de un objeto-mantequilla es también un objeto-mantequilla:

$$x \in Mantequilla \wedge ParteDe(y, x) \Rightarrow y \in Mantequilla$$

Se puede decir ahora, que la mantequilla se derrite aproximadamente a los 30 °C:

$$x \in Mantequilla \Rightarrow PuntoDeFusión(x, Centígrados(30))$$

La mantequilla es amarilla, menos densa que el agua, se reblandece a temperatura ambiente, tiene alto contenido de grasas, etc. Por otra parte, la mantequilla no tiene tamaño, peso, ni forma específicos. Lo que sí se puede es definir categorías más especializadas para la mantequilla, por ejemplo *MantequillaSinSal*, que también es un tipo de sustancia, puesto que una parte de un objeto-mantequilla-sin-sal es también un objeto-mantequilla-sin-sal. Por otra parte, si se define una categoría *KiloDeMantequilla*, cuyos miembros sean todos los objetos-mantequilla que pesen un kilo, ¡ya se tiene una sustancia! Si se parte un kilo de mantequilla por la mitad, muy a pesar nuestro, el resultado no serán dos kilos de mantequilla.

INTRÍNSECAS

Lo que realmente ha sucedido es lo siguiente: hay propiedades que son **intrínsecas**, pertenecen a la misma sustancia del objeto más que el objeto como un todo. Cuando se divide algo en dos, ambas partes conservan el mismo conjunto de propiedades

EXTRÍNSECAS

intrínsecas (cosas como la densidad, el punto de ebullición, el sabor, el color, la propiedad, etc.). Por otra parte, las propiedades **extrínsecas** son lo contrario: propiedades tales como peso, longitud, forma, función, etc., que después de dividir algo no se conservan.

Aquella clase de objetos que en su definición incorpore sólo propiedades *intrínsecas* es una sustancia, o un sustantivo no contable, mientras que una clase que incorpore *cualquier* propiedad extrínseca en su definición es un sustantivo contable. La categoría *Sustancia* es la categoría más general de las sustancias, en la que no se especifica ninguna propiedad intrínseca. La categoría *Objeto*, es la categoría de objetos discretos más general, en la que no se especifica ninguna propiedad extrínseca.

10.3 Acciones, situaciones y eventos

El razonamiento sobre los resultados de las acciones es fundamental para el funcionamiento de un agente basado en conocimiento. El Capítulo 7 proporciona ejemplos de sentencias proposicionales que describen cómo las acciones afectan al mundo de *wumpus* (por ejemplo, la Ecuación (7.3) en el Apartado 7.7, describe cómo la posición del agente cambia debido a un movimiento delantero). Una desventaja de la lógica proposicional, es la necesidad de tener diferentes copias de la descripción de la acción para cada intervalo de tiempo en la cual la acción se podría llevar a cabo. Esta sección describe un método de representación que utiliza lógica de primer orden para resolver este problema.

La ontología del cálculo de situaciones

Una forma obvia de resolver la necesidad de disponer de múltiples copias de los axiomas es simplemente cuantificar en el tiempo ($\forall\ t$, tal que es el resultado en $t + 1$ de realizar la acción en t). En vez de tratar con intervalos de tiempo explícitos como $t + 1$, esta sección utiliza *situaciones*, que denotan los estados resultantes de ejecutar acciones. Esta aproximación se denomina **cálculo de situaciones** y utiliza las siguientes ontologías:

CÁLCULO DE SITUACIONES

- Como en el Capítulo 8, las acciones son términos lógicos como *HaciaDelante* y *Girar*(*derecha*). Por ahora, se asumirá que el entorno contiene sólo un agente (si existe más de uno, se debe insertar un argumento adicional para decir qué agente está realizando las acciones).

SITUACIONES

- Las **situaciones** son términos lógicos que consisten en una situación inicial (normalmente denominada S_0), y todas las situaciones que son generadas mediante la aplicación de una acción a una situación. La función *Resultado*(a, s) (en ocasiones denominada *Do*), da nombre a la situación resultante de ejecutar una acción a en una situación s. La Figura 10.2 ilustra esta idea.

FLUJOS

- Los **flujos** son funciones y predicados que varían de una situación a la siguiente, como la posición de un agente o la vitalidad de *wumpus*. El diccionario dice que flujo representa algo que fluye, como un líquido. Utilizando este concepto, se quie-

Figura 10.2 En el cálculo de situaciones, cada situación (excepto S_0) es el resultado de una acción.

re significar el flujo o cambio a través de las situaciones. Por convención, la situación es siempre el último argumento de un flujo. Por ejemplo, $\neg Sostener(G_1, S_0)$ indica que el agente no está sosteniendo el lingote de oro G_1 en la situación inicial S_0. $Edad(Wumpus, S_0)$ se refiere a la edad de *wumpus* en S_0.

- Las funciones o predicados **atemporales** o **eternos** también se permiten. Ejemplos son el predicado $Lingote_de_oro(G_1)$ y la función $PiernaIzquierdaDe(Wumpus)$.

Como complemento a las acciones simples, también es útil razonar sobre las secuencias de acciones. Se pueden definir los resultados de las secuencias en términos de los resultados de acciones individuales. Primero, se establecerá que ejecutar una secuencia vacía, deja la situación inalterada

$$Resultado([\ ], s) = s$$

Ejecutar una secuencia no vacía es lo mismo que ejecutar la primera acción, y entonces ejecutar el resto sobre la situación resultante:

$$Resultado([a]\ seq], s) = Resultado(seq, Resultado(a, s))$$

PROYECCIÓN

PLANIFICACIÓN

Un agente de cálculo de situaciones debería ser capaz de deducir el resultado de una secuencia dada de acciones. Esta es la tarea de la **proyección**. Con un algoritmo adecuado de inferencia constructiva, debería ser capaz de encontrar una secuencia que logre el efecto deseado. Esto es lo que se denomina tarea de **planificación**.

Se utilizará un ejemplo de una versión modificada del mundo de *wumpus*, en la que no se tiene en cuenta la orientación del agente y a dónde puede *Ir*, dada una localiza-

ción y su localización adyacente. Supóngase que el agente está en [1, 1] y que el lingote de oro está en [1, 2]. El objetivo es tener el lingote de oro en [1, 1]. Los predicados de flujo son $En(o, x, s)$ y $Sosteniendo(o, s)$. Entonces la base de conocimiento inicial podría incluir la siguiente descripción:

$$En(Agente, [1, 1], S_0) \wedge En(G1, [1, 2], S_0)$$

De todas formas esto no es suficiente, porque no especifica qué no es cierto en S_0 (para una discusión más en profundidad sobre este punto, *véase* el Apartado 10.7). La descripción completa sería como sigue:

$$En(o, x, S_0) \Leftrightarrow [(o = Agente \wedge x = [1, 1]) \vee (o = G_1 \wedge x = [1, 2])]$$
$$\neg Sosteniendo(o, S_0)$$

También se necesita decir que G_1 es un lingote de oro y que [1, 1] y [1, 2] son adyacentes:

$$Lingote_de_oro(G_1) \wedge Adyacente([1, 1], [1, 2]) \wedge Adyacente([1, 2], [1, 1])$$

Uno desearía ser capaz de probar que el agente consigue su objetivo desplazándose a [1, 2], cogiendo el lingote de oro y volviendo a [1, 1]. Es decir:

$$En(G_1, [1, 1], Resultado([Ir([1, 1], [1, 2]), Tomar(G_1), Ir([1, 2], [1, 1])], S_0))$$

Una posibilidad más interesante es la de construir un plan para tomar el lingote de oro respondiendo a la pregunta «¿qué secuencia de acciones tienen como resultado que el lingote de oro esté al final en la posición [1, 1]?».

$$\exists\ seq\ En(G_1, [1, 1], Resultado(seq, S_0))$$

Se verá qué debe existir en la base de conocimiento para que preguntas como ésta se puedan responder.

Descripción de acciones en el cálculo de situaciones

AXIOMA DE POSIBILIDAD

AXIOMA DE EFECTO

En la versión más simple del cálculo de situaciones, cada acción se describe por dos axiomas: un **axioma de posibilidad** que especifica cuándo es posible ejecutar una acción, y un **axioma de efecto** que determina qué sucede cuando se ejecuta una acción posible. Se utilizará $Posible(a, s)$ para expresar que es posible la ejecución de la acción a en la situación s. Los axiomas tienen la siguiente forma:

AXIOMA DE POSIBILIDAD: $Precondiciones \Rightarrow Posible(a, s)$.
AXIOMA DE EFECTO: $Posible(a, s) \Rightarrow$ *Cambios que son el resultado de ejecutar una acción*.

Se presentan estos axiomas para el mundo modificado de *wumpus*. Para hacer más cortas las sentencias, se omitirán los cuantificadores universales cuyo ámbito sea la sentencia entera. Se asumirá que la variable s representa situaciones, a representa acciones, o representa objetos (incluyendo agentes), g representa lingotes de oro y x e y representan localizaciones.

Los axiomas de posibilidad para este mundo establecen que un agente puede moverse entre localizaciones adyacentes, tomar un lingote de oro en la posición actual y soltar un lingote de oro que está sosteniendo:

$$\begin{array}{lcl} En(Agente, x, s) \wedge Adyacente(x, y) & \Rightarrow & Posible(Ir(x, y), s) \\ Lingote_de_oro(g) \wedge En(Agente, x, s) \wedge En(g, x, s) & \Rightarrow & Posible(Tomar(g), s) \\ Sosteniendo(g, s) & \Rightarrow & Posible(Soltar(g), s) \end{array}$$

Los axiomas de efecto establecen que, si una acción es posible, entonces ciertas propiedades (flujos) tendrán lugar en la situación resultante de ejecutar la acción. Ir desde x a y supone estar en y, tomar el lingote de oro conlleva sostenerlo y soltar el lingote de oro supone no sostenerlo:

$$\begin{array}{lcl} Posible(Ir(x, y), s) & \Rightarrow & En(Agente, y, Resultado(Ir(x, y), s)) \\ Posible(Tomar(g), s) & \Rightarrow & Sosteniendo(g, Resultado(Tomar(g), s)) \\ Posible(Soltar(g), s) & \Rightarrow & \neg Sosteniendo(g, Resultado(Soltar(g), s)) \end{array}$$

Al haber propuesto estos axiomas, ¿se puede probar que este pequeño plan alcanzará el objetivo?, ¡desafortunadamente no! Al principio todo funciona correctamente: $Ir([1, 1], [1, 2])$ es ciertamente posible en S_0 y el axioma de efecto para Ir permite concluir que el agente alcanza [1, 2]:

$$En(Agente, [1, 2], Resultado(Ir([1, 1], [1, 2]), S_0))$$

Ahora se considerará la acción $Tomar(G_1)$. Se debe mostrar que en la nueva situación es posible, es decir,

$$En(G_1, [1, 2], Resultado(Ir([1\ 1], [1,2]), S_0))$$

Pero desgraciadamente nada en la base de conocimiento justifica esta conclusión. Intuitivamente, se entiende que la acción del agente Ir no debería de tener efecto en la colocación del lingote de oro, por lo tanto, éste debería estar en la posición [1, 2], donde estaba en la situación S_0. *El problema es que el axioma de efecto dice lo que cambia, pero no lo que permanece igual.*

PROBLEMA DEL MARCO

Representar todas las cosas que permanecen inalterables es lo que se conoce con el nombre del **problema del marco**[2]. Se debe encontrar una solución eficiente al problema del marco porque en el mundo real, casi todo permanece inalterable todo el tiempo. Cada acción afecta sólo a una pequeña fracción de todo lo que fluye.

AXIOMAS MARCO

Una aproximación es escribir **axiomas marco** explícitos, que lo que hagan sea especificar qué permanece inalterable. Por ejemplo, los movimientos del agente dejan otros objetos en la misma posición a menos que sean tomados:

$$En(o, x, s) \wedge (o \neq Agente) \wedge \neg Sosteniendo(o, s) \Rightarrow En(o, x, Resultado(Ir(y, z), s))$$

Si hay F predicados de flujo y A acciones, entonces se necesitarán $O(AF)$ axiomas marco. Por otro lado, si cada acción tiene como mucho E efectos, donde normalmente E es mucho menor que F, entonces se podría representar lo que sucede con una base de co-

[2] El nombre de «problema del marco» procede del concepto físico de «marco de referencia» (el fondo que se asume fijo respecto al cual se mide la acción). También tiene relación con el fondo de una película, en el cual se producen pocos cambios de una imagen a otra.

PROBLEMA DE LA REPRESENTACIÓN DEL MARCO

PROBLEMA DE LA INFERENCIA DEL MARCO

nocimiento mucho menor de tamaño $O(AE)$. Este es el **problema de la representación del marco**. El problema cercano relacionado es el **problema de la inferencia del marco,** consistente en proyectar los resultados de una secuencia de acciones de t fases en el instante de tiempo $O(Et)$, en lugar de en el instante $O(Ft)$ o $O(AEt)$. Se abordará cada problema por separado.

Resolver el problema de la representación del marco

AXIOMAS ESTADO-SUCESOR

La solución al problema de la representación del marco implica un ligero cambio en el punto de vista utilizado para escribir los axiomas. En lugar de especificar los efectos de cada acción, se considerará la forma en que cada predicado de flujo evoluciona en el tiempo[3]. Se denominará a los axiomas a utilizar **axiomas estado-sucesor**. Estos axiomas tienen la siguiente forma:

AXIOMA ESTADO-SUCESOR:

Acción es Posible $\Rightarrow$
(*Flujo es cierto en el estado resultante* $\Leftrightarrow$ *Los efectos de las acciones se produjeron*
$\vee$ *Eran ciertos antes y la acción los dejó igual*)

Después de la salvedad de que no se consideran acciones imposibles, nótese que la definición utiliza $\Leftrightarrow$, no $\Rightarrow$. Esto significa que el axioma especifica que el flujo será cierto si *y sólo si* la parte derecha es cierta. Dicho de otra forma, se especifica que el valor de verdad de cada flujo en el siguiente estado es una función de la acción y del valor de verdad en el estado actual. Esto significa que el siguiente estado viene especificado de forma completa por el estado actual, y por lo tanto, no se necesitan axiomas marco adicionales.

El axioma estado-sucesor para la localización del agente dice que el agente está en y después de ejecutar una acción, bien porque la acción es posible y consiste en el movimiento a y, o bien porque el agente se encontraba en y y la acción no es un movimiento a ningún lado:

$$\begin{aligned} &Posible(a, s) \Rightarrow \\ &\quad (En(Agente, y, Resultado(a, s)) \quad \Leftrightarrow \quad a = Ir(x, y) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \vee (En(Agente, y, s) \wedge a \neq Ir(y, z))) \end{aligned}$$

El axioma para *Sosteniendo* dice que el agente está sosteniendo g después de ejecutar una acción, siempre que la acción fuera tomar aplicado a g suponiendo que la acción tomar es posible, o bien si el agente ya estaba sosteniendo g y la acción no supone soltarlo:

$$\begin{aligned} &Posible(a, s) \Rightarrow \\ &\quad (Sosteniendo(g, Resultado(a, s)) \quad \Leftrightarrow \quad a = Tomar(\mathrm{g}) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \vee (Sosteniendo(g, s) \wedge a \neq Soltar(g))) \end{aligned}$$

[3] Esta es esencialmente la aproximación que utilizamos en la construcción del agente basado en circuito booleano del Capítulo 7. De hecho, los axiomas como las Ecuaciones (7.4) y (7.5) pueden ser vistos como axiomas estado-sucesor.

Los axiomas estado-sucesor solucionan el problema de la representación del marco, porque el tamaño total de axiomas es $O(AE)$ literales: cada uno de los E efectos de cada acción A, se menciona exactamente una vez. Los literales se aplican sobre los F diferentes axiomas, por lo que los axiomas tienen un tamaño medio de AE/F.

El lector inteligente habrá notado que los axiomas manejan el flujo *En* para el agente, pero no para el lingote de oro. Por lo tanto, no se puede demostrar todavía que el plan de tres fases consiga el objetivo de tener el lingote de oro en [1, 1]. Se necesita exponer que un **efecto implícito** de que un agente se mueva desde una posición x a una posición y, es que cualquier lingote de oro que porte, se moverá también (así como cualquier hormiga que estuviera en el lingote, cualquier bacteria en la hormiga, etc.). El tratar con efectos implícitos se conoce como el **problema de la ramificación**. Se discutirá el problema en general posteriormente, pero para este dominio específico se puede resolver escribiendo un axioma estado-sucesor más general para *En*. El nuevo axioma que engloba la versión anterior establece que un objeto o está en la posición y si el agente fue a y y o es el agente o algo que el agente estaba sosteniendo, o si o se encontraba ya en la posición y y el agente no fue a ningún otro sitio, siendo o el agente o algo que el agente estaba sosteniendo.

EFECTO IMPLÍCITO

PROBLEMA DE LA RAMIFICACIÓN

$$
\begin{aligned}
&Posible(a, s) \Rightarrow \\
&(En(o, y, Resultado(a, s)) \Leftrightarrow (a = Ir(x, y) \wedge (o = Agente \vee Sosteniendo(o, s))) \\
&\qquad \vee (En(o, y, s) \wedge \neg(\exists\, z\ \ y \neq z \wedge a = Ir(y, z) \wedge \\
&\qquad\qquad (o = Agente \vee Sosteniendo(o, s)))).
\end{aligned}
$$

Existe un tecnicismo más: un proceso de inferencia que use estos axiomas debe ser capaz de evaluar desigualdades. El tipo de desigualdad más sencilla es entre constantes (por ejemplo, $Agente \neq G_1$). La semántica general de la lógica de primer orden permite distinguir constantes para referirse al mismo objeto, por lo tanto, la base de conocimiento debe incluir un axioma para prevenir esto. Los **axiomas de nombre único** establecen una desigualdad para cada par de constantes en la base de conocimiento. Cuando esto se asume por el demostrador de teoremas en vez de ser especificado en la base de conocimiento, se denomina **asunción de nombres únicos**. También se necesitan especificar desigualdades entre los términos de las acciones: $Ir([1, 1], [1, 2])$ es una acción diferente a $Ir([1, 2], [1, 1])$ o $Tomar(G_1)$. Primero, se establece que cada tipo de acción es distinta (que la acción *Ir* no es una acción *Tomar*). Para cada par de nombres de acción A y B, se tendrá que

AXIOMAS DE NOMBRE ÚNICO

$$A(x_1..., x_m) \neq B(y_1..., y_n)$$

A continuación, se establece que dos términos de acción con el mismo nombre de acción se refieren a la misma acción, sólo si las acciones afectan a los mismos objetos:

$$A(x_1..., x_m) = A(y_1..., y_n) \Leftrightarrow x_1 = y_1 \wedge ... \wedge x_m = y_m$$

Todo esto se denomina conjuntamente **axiomas de acción única**. La combinación de la descripción del estado inicial, axiomas estado-sucesor, nombres de axioma únicos y axiomas de acción única, es suficiente para demostrar que el plan propuesto consigue el objetivo.

AXIOMAS DE ACCIÓN ÚNICA

Resolver el problema de la inferencia del marco

Los axiomas estado-sucesor resuelven el problema de la representación del marco, pero no el problema de la inferencia del marco. Considérese un plan p de t fases en el que $S_t = Resultado(p, S_0)$. Para decidir qué flujos son ciertos en S_t, se necesita considerar cada uno de los axiomas marco F en cada uno de las fases de tiempo t. Debido a que los axiomas tienen un tamaño medio de AE/F, esto supone un trabajo de inferencia de $O(AEt)$. La mayor parte del trabajo corresponde a copiar flujos que no cambian de una situación a la siguiente.

Para resolver el problema de la inferencia del marco, existen dos posibilidades. Primeramente, se podría descartar el cálculo de las situaciones e inventar un nuevo formalismo para escribir axiomas. Esto ha sido llevado a cabo por formalismos como el **cálculo de flujos**. En segundo lugar, se podría alterar el mecanismo de inferencia para manejar axiomas marco de forma más eficiente. Un detalle que debería ser posible es que la aproximación más simple fuera $O(AEt)$. ¿Por qué debería depender del número de acciones, A, cuando se conoce que se ejecuta exactamente una acción en cada instante de tiempo? Para ver cómo se pueden mejorar las cosas, primero se presentará el formato de los axiomas marco:

$$\begin{aligned} &Posible(a, s) \Rightarrow \\ &\quad F_i(Resultado(a, s)) \quad \Leftrightarrow \quad (a = A_1 \vee a = A_2 \ldots) \\ &\qquad\qquad\qquad\qquad \vee F_i(s) \wedge (a \neq A_3) \wedge (a \neq A_4) \wedge \ldots \end{aligned}$$

Es decir, cada axioma menciona varias acciones que pueden hacer el flujo cierto y varias acciones que pueden hacerlo falso. Esto se puede formalizar introduciendo el predicado $EfectoPos(a, F_i)$, que significa que una acción a hace que F_i sea cierto, y $EfectoNeg(a, F_i)$ que significa que a hace que F_i sea falso. Entonces se puede rescribir el esquema de acciones anterior como:

$$\begin{aligned} &Posible(a, s) \Rightarrow \\ &\quad F_i(Resultado(a, s)) \quad \Leftrightarrow \quad EfectoPos(a, F_i) \vee [F_i(s) \wedge \neg EfectoNeg(a, F_i)] \\ &EfectoPos(A_1, F_i) \\ &EfectoPos(A_2, F_i) \\ &EfectoNeg(A_3, F_i) \\ &EfectoNeg(A_4, F_i) \end{aligned}$$

Que esto se pueda hacer automáticamente, depende del formato exacto de los axiomas marco. Para llevar a cabo un procedimiento de inferencia eficiente usando axiomas como éste, se necesita realizar lo siguiente:

1. Indexar los predicados *EfectoPos* y *EfectoNeg* por su primer argumento, de forma que cuando se tenga una acción que ocurre en el instante de tiempo t, se pueda encontrar su efecto en un tiempo de $O(1)$.
2. Indexar los axiomas de tal forma que cuando se conozca que F_i es un efecto de una acción, se pueda encontrar el axioma para F_i en un tiempo de $O(1)$. Por lo tanto, no es necesario considerar los axiomas para flujos que no son un efecto de una acción.

3. Representar cada situación como una situación previa más un incrementeo. De este modo, si no cambia nada desde un paso al siguiente, no se necesita realizar trabajo alguno. En la aproximación antigua, se necesitaría hacer un trabajo de $O(F)$ para generar una sentencia para cada flujo $F_i(Resultado(a, s))$ de la sentencia $F_i(s)$ precedente.

Por lo tanto, en cada instante de tiempo, es necesario centrarse en la acción actual, buscando sus efectos y actualizando el conjunto de flujos ciertos. Cada instante de tiempo tendrá una media E de estas actualizaciones, con una complejidad total de $O(Et)$. Esto constituye una solución al problema de la inferencia del marco.

El tiempo y el cálculo de eventos

El cálculo de situaciones funciona bien cuando existe un agente simple realizando acciones discretas e instantáneas. Cuando las acciones tienen una duración y se pueden solapar unas con otras, el cálculo de situaciones se convierte en engorroso. Por lo tanto, estos temas se tratarán con un formalismo alternativo conocido como el **cálculo de eventos**, basado en puntos en el tiempo en vez de en situaciones. Los términos «evento» y «acción» se pueden intercambiar. Informalmente, un «evento» se corresponde con un conjunto amplio de acciones, incluyendo aquellas sin un agente explícito. Son más sencillas de manejar en el cálculo de eventos que en el cálculo de situaciones.

CÁLCULO DE EVENTOS

En el cálculo de eventos, los flujos tienen lugar en puntos en el tiempo en vez de en situaciones. El axioma de cálculo de eventos dice que un flujo es cierto en un punto concreto en el tiempo, si el flujo fue iniciado por un evento en un instante de tiempo anterior y no fue finalizado por la intervención de algún otro evento. Las relaciones *Inicio* y *Terminación* representan un papel similar al de la relación *Resultado* en el cálculo de situaciones. $Inicio(e, f, t)$ significa que la ocurrencia del evento e en el tiempo t causa que el flujo f sea cierto, mientras que $Terminación(w, f, t)$ significa que f deja de ser cierto. Se utilizará $Sucede(e, t)$ para reflejar que el evento e sucede en el tiempo t, y se utilizará $Interrumpido(f, t, t_2)$ para expresar que f ha finalizado por algún evento en algún instante entre t y t_2. Formalmente, el axioma es:

AXIOMA DE CÁLCULO DE EVENTOS:

$$T(f, t_2) \Leftrightarrow \exists\, e, t \;\; Sucede(e, t) \wedge Inicio(e, f, t) \wedge (t < t_2) \;\; \wedge \neg Interrumpido(f, t, t_2)$$

$$Interrumpido(f, t, t_2) \Leftrightarrow \exists\, e, t_1 \;\; Sucede(e, t_1) \wedge Terminación(e, f, t_1) \wedge (t < t_1) \wedge (t_1 < t_2)$$

Esto proporciona una funcionalidad similar al cálculo de situaciones, pero con la habilidad de poder hablar de puntos en el tiempo e intervalos; por lo tanto, se puede afirmar $Sucede(Apagar(Conmutador_1), 1{:}00)$ para decir que una llave de luz será apagada exactamente a la 1:00 h.

Se han hecho varias extensiones al cálculo de eventos (algunas con más éxito que otras), para solucionar los problemas derivados de tener que representar eventos con duración, eventos concurrentes, eventos que cambian continuamente y otras complicaciones.

El cálculo de eventos puede ser extendido para manejar efectos indirectos, cambios continuos, efectos no deterministas, restricciones causales y otras situaciones. Se retomarán algunos de estos temas en el Apartado 10.3.

Eventos generalizados

Hasta ahora se han revisado dos conceptos principales: acciones y objetos. Ahora es el momento de ver cómo encajan en una ontología global, en la cual las acciones y los objetos pueden ser vistos como aspectos de un universo físico. Se utilizará un universo particular compuesto por dimensiones espacial y temporal. En el mundo de *wumpus*, el componente espacial se corresponde con una rejilla bidimensional y el tiempo es discreto. El mundo real tiene tres dimensiones en el espacio y una dimensión en el tiempo[4], todas continuas. Un **evento generalizado** se compone de aspectos de alguna pieza espacio-temporal (un segmento del universo espacio-temporal de múltiples dimensiones). Esta abstracción generaliza la mayoría de los conceptos que se han visto hasta ahora, incluyendo las acciones, localizaciones, tiempo, flujos y objetos físicos. La Figura 10.3 da una idea general. A partir de ahora, se utilizará el término simple «evento» para referirse a eventos generalizados.

EVENTO GENERALIZADO

Por ejemplo, La Segunda Guerra Mundial es un evento que tuvo lugar en varios puntos en el espacio-tiempo, como representa la zona sombreada de la figura. Puede ser dividida en **subeventos**[5]:

SUBEVENTOS

$$SubEvento(BatallaDeBretaña, SegundaGuerraMundial)$$

De modo similar, la Segunda Guerra Mundial es un subevento del siglo XX:

$$SubEvento(SegundaGuerraMundial, SigloXX)$$

El siglo XX es un *intervalo* de tiempo. Los intervalos son trozos de especio-tiempo que incluyen todo el espacio entre dos puntos de tiempo. La función *Período*(*e*) denota el intervalo más pequeño que encierra al evento *e*. *Duración*(*i*) es la longitud del tiempo que ocupa un intervalo, por lo tanto se puede decir *Duración*(*Período*(*SegundaGuerraMundial*)) $>$ *Años*(5).

Australia es un lugar. Un trozo con unos bordes delimitados en el espacio. Los bordes pueden variar en el tiempo, debido a cambios geológicos o políticos. Se utiliza el predicado *En* para denotar la relación de subevento que tiene lugar cuando la proyección espacial de un evento es *ParteDe* otro:

$$En(Sydney, Australia)$$

La función *Localización*(*e*) denota el lugar más pequeño que encierra al evento *e*.

Como cualquier otro tipo de objetos, los eventos se pueden agrupar en categorías. Por ejemplo, *SegundaGuerraMundial* pertenece a la categoría de *Guerras*. Para decir que una guerra civil ocurrió en Inglaterra en 1640, se podría decir:

$$\exists w \;\; w \in GuerrasCiviles \wedge SubEvento(w, 1640) \wedge En(Localización(w), Inglaterra)$$

La noción de categoría de eventos responde a una pregunta que se obvió cuando se describieron los efectos de los axiomas en el Apartado 10.3: ¿A qué se refieren los térmi-

[4] Algunos físicos que estudian la teoría de la cadena, hablan de 10 dimensiones o más, y algunos hablan de un mundo discreto, pero una representación espacio-temporal continua de cuatro dimensiones es adecuada para el propósito de hacer razonamientos basados en el sentido común.

[5] Nótese que *SubEvento* es un caso especial de la relación *ParteDe*, siendo también transitiva y reflexiva.

Figura 10.3 Eventos generalizados. Un universo tiene dimensiones espaciales y temporales. En esta figura se muestra sólo una dimensión espacial simple. Todos los eventos son *ParteDe* el universo. Un evento, como la *SegundaGuerraMundial*, ocurre en una porción de espacio-tiempo de forma algo arbitraria y con variación en los límites temporales. Un *Intervalo*, como el *SigloXX*, tiene una extensión temporal fija y una extensión espacial máxima, mientras que un *Lugar*, como Australia, tiene una extensión espacial fija y una extensión temporal máxima.

nos lógicos como $Ir([1, 1], [1, 2])$?, ¿son eventos? La respuesta, posiblemente sorprendente, es *no*. Se puede entender esto considerando un plan con dos acciones «idénticas» como

$$[Ir([1, 1], [1, 2]), Ir([1, 2], [1, 1]), Ir([1, 1], [1, 2])]$$

En este plan, $Ir([1, 1], [1, 2])$ no puede ser el nombre de un evento porque hay *dos eventos diferentes* que ocurren en tiempos diferentes. En su lugar, $Ir([1, 1], [1, 2])$ es el nombre de una *categoría* de eventos (representando todos los eventos mediante los cuales el agente va desde [1, 1] a [1, 2]). El plan de tres fases establece que ocurrirán instancias de estos tres eventos.

Nótese que esta es la primera vez que se han visto nombres de categorías formados por términos complejos en vez de por constantes. Esto no presenta nuevas dificultados. De hecho, se puede usar la estructura de los argumentos como una ventaja. Mediante la eliminación de argumentos se crea una categoría más general.

$$Ir(x, y) \subseteq IrA(y) \qquad\qquad Ir(x, y) \subseteq IrDesde(x)$$

De modo similar, se pueden añadir argumentos para crear categorías más específicas. Por ejemplo, para describir acciones de otros agentes, se puede añadir un argumento que represente al agente. Por lo tanto, el decir que Shankar voló ayer desde Nueva York a Nueva Delhi, se podría escribir:

$$\exists\, e \;\; e \in Volar(Shankar, NuevaYork, NuevaDelhi) \wedge SubEvento(e, Ayer)$$

La forma de esta fórmula es tan común que se creará una abreviación para ella: $E(c, i)$ significará que un elemento de una categoría de eventos c es un subevento del evento o intervalo i:

$$E(c, i) \Leftrightarrow \exists\, e \;\; e \in c \;\wedge SubEvento(e, i)$$

Por lo tanto tenemos que,

$$E(Volar(Shankar, NuevaYork, NuevaDelhi), Ayer)$$

Procesos

EVENTOS DISCRETOS

Los eventos que se han visto hasta ahora son los que se denominan **eventos discretos** (con una estructura definida). El viaje de Shankar tiene un comienzo, un punto intermedio y un final. Si se interrumpe a medio camino, el evento será diferente (no será un viaje desde Nueva York a Nueva Delhi, sino un viaje desde Nueva York a algún lugar en Europa). Por otro lado, la categoría de eventos representados por *Volando*(*Shankar*) tiene una cualidad diferente. Si se considera un pequeño intervalo en el vuelo de Shankar, por ejemplo el tercer segmento de 20 minutos (mientras que él espera ansiosamente por una segunda maleta de cacahuetes), este evento forma parte todavía de *Volando*(*Shankar*). De hecho, esto es cierto para cualquier subintervalo.

PROCESOS

EVENTOS LÍQUIDOS

Las categorías de eventos que cumplen esta propiedad se llaman categorías de **procesos** o categorías de **eventos líquidos**. Todos los subintervalos de un proceso son también miembros de la misma categoría de procesos. Mediante la misma notación que se empleó en los eventos discretos, se puede afirmar que, por ejemplo, Shankar iba volando en algún momento del día de ayer:

$$E(Volando(Shankar), Ayer)$$

Frecuentemente se necesitará expresar que algún proceso se realizó *durante* cierto intervalo, en vez de que sólo se realizó durante cierto subintervalo. Para ello, utilizamos el predicado T:

$$T(Trabajando(Stuart), HoyHoraDelAlmuerzo)$$

$T(c, i)$ significa que cierto evento de tipo c se produjo exactamente durante el intervalo i, es decir, el evento comienza y termina al mismo tiempo que el intervalo.

SUSTANCIAS TEMPORALES

SUSTANCIAS ESPACIALES

La diferencia entre eventos líquidos y no líquidos es análoga a la diferencia entre sustancias o materia, y objetos individuales. De hecho, algunos han llamado **sustancias temporales** a los eventos líquidos, mientras que cosas como la mantequilla son **sustancias espaciales**.

ESTADOS

De la misma forma que se describen los procesos de cambio continuo, los eventos líquidos puede describir procesos de cambio no continuo. Son llamados **estados**. Por ejemplo, «Encontrándose Shankar en Nueva York» es una categoría de estados que se denotan como *En*(*Shankar, NuevaYork*). Para decir que él estuvo en Nueva York todo el día, se escribiría

$$T(En(Shankar, NuevaYork), Hoy)$$

CÁLCULO DE FLUJOS

Se pueden formar eventos y estados más complejos combinando las primitivas. Esta aproximación recibe el nombre de **cálculo de flujos**. El cálculo de flujos se refiere a la combinación de flujos, no a flujos individuales. Ya se ha visto una forma de representar el evento correspondiente al momento en que dos cosas suceden al mismo tiempo, denominado $Ambos(e_1, e_2)$. En cálculo de flujos, esto se abrevia normalmente con la notación infija $e_1 \circ e_2$. Por ejemplo, para decir que alguien caminó y masticaba chicle al mismo tiempo, se puede escribir

$$\exists\, p, i \;\; (p \in Gente) \wedge T(Caminar(p) \circ MasticarChicle(p), i)$$

La función «$\circ$» es conmutativa y asociativa, como la conjunción lógica. Se pueden definir funciones análogas a las de disyunción y negación, pero se debe tener cuidado (hay dos formas razonables de interpretar la disyunción). Cuando se dice «el agente o estaba caminando o estaba masticando chicle durante los dos últimos minutos» se puede querer decir que el agente estaba haciendo una de las dos acciones durante todo el intervalo de tiempo, o quizá que estaba alternando entre las dos acciones. Se utilizará *UnaDe* y *Cualquiera* para indicar estas dos posibilidades. La Figura 10.4 representa los eventos complejos.

Figura 10.4 Una representación de eventos complejos. (a) $T(Ambos(p, q), i)$, también denotado como $T(p \circ q, i)$, (b) $T(UnoDe(p, q), i)$ (c) $T(Cualquiera(p, q), i)$.

Intervalos

El tiempo es importante para cualquier agente que realice acciones, y se ha realizado mucho trabajo para la representación de intervalos de tiempo. Aquí se consideran dos clases: momentos e intervalos extendidos. La diferencia es que sólo los momentos tienen duración cero:

$$Partición(\{Momentos, IntervalosExtendidos\}, Intervalos)$$
$$i \in Momentos \Leftrightarrow Duración(i) = Segundos(0)$$

A continuación se creará una escala de tiempo y se asociarán puntos con momentos en esa escala, de lo cual se obtiene el concepto de tiempo absoluto. La escala de tiempo es arbitraria. Se medirá en segundos y se dirá que el momento de medianoche (GMT) del 1 de enero de 1900 tiene un valor de tiempo igual a cero. Las funciones *Comienzo* y *Fin* seleccionan los momentos más tempranos y tardíos en un intervalo, y la función *Tiempo* determina el punto en la escala de tiempo para un momento. La función *Duración* devuelve la diferencia entre el momento final y el inicial.

$$Intervalo(i) \Rightarrow Duración(i) = (Tiempo(Fin(i)) - (Tiempo(Comienzo(i)))$$
$$Tiempo(Comienzo(1900DC)) = Segundos(0)$$
$$Tiempo(Comienzo(2001DC)) = Segundos(3187324800)$$
$$Tiempo(Fin(2001DC)) = Segundos(3218860800)$$
$$Duración(2001DC) = Segundos(31536000)$$

Para facilitar la lectura de los números, también se introduce una función *Fecha*, la cual recibe seis argumentos (horas, minutos, segundos, día, mes y año) y devuelve un punto en el tiempo:

$$Tiempo(Fecha(2001DC)) = Fecha(0, 0, 0, 1, Ene, 2001)$$
$$Fecha(0, 20, 21, 24, 1, 1995) = Segundos(3000000000)$$

Dos intervalos *Coinciden* si el momento final del primero es igual al momento de inicio del segundo. También es posible definir predicados como *Antes*, *Después*, *Durante* y *Solapamiento* únicamente en términos de *Coinciden*, pero es más intuitivo definirlos en términos de puntos en la escala de tiempo (*véase* la Figura 10.5 para una representación gráfica).

$$\begin{array}{lcl}
Coinciden(i, j) & \Leftrightarrow & Tiempo(Fin(i)) = Tiempo(Inicio(j)) \\
Antes(i, j) & \Leftrightarrow & Tiempo(Fin(i)) < Tiempo(Inicio(j)) \\
Después(j, i) & \Leftrightarrow & Antes(i, j) \\
Durante(i, j) & \Leftrightarrow & Tiempo(Inicio(j)) \leq Tiempo(Inicio(i)) \\
 & & \wedge\, Tiempo(Fin(i)) \leq Tiempo(Fin(j)) \\
Solapamiento(i, j) & \Leftrightarrow & \exists\, k\;\; Durante(k, i) \wedge Durante(k, j)
\end{array}$$

Por ejemplo, para decir que el reinado de Isabel II siguió al de Jorge VI, y el reinado de Elvis se solapó con la década de los 50, se puede escribir lo siguiente:

$$Antes(ReinadoDe(IsabelII), ReinadoDe(JorgeVI))$$
$$Solapamiento(AñosCincuenta, ReinadoDe(Elvis))$$
$$Inicio(AñosCincuenta) = Inicio(1950DC)$$
$$Fin(AñosCincuenta) = Fin(1959DC)$$

Figura 10.5 Predicados sobre intervalos de tiempo.

Flujos y objetos

Se ha mencionado que los objetos físicos pueden ser vistos como eventos generalizados, en el sentido de que un objeto físico es un trozo de espacio-tiempo. Por ejemplo, Estados Unidos puede ser visto como un evento que empezó, digamos que en 1776 con la unión de los 13 estados, y que se encuentra hoy en día en progreso como unión de 50 estados. Se puede describir el cambio en las propiedades de Estados Unidos usando estados de flujo. Por ejemplo, se puede decir que en un momento determinado de 1999, la población ascendía a 271 millones:

$$E(Población(EEUU, 271000000), 1999DC)$$

Otra propiedad de Estados Unidos que cambia cada cuatro u ocho años, a menos que no suceda algún contratiempo, es el presidente. Se podría proponer que el término lógico *Presidente*(*EEUU*) denotara un objeto diferente para distintos momentos en el tiempo. Desgraciadamente esto no es posible, porque un término denota exactamente a un objeto en una estructura de modelos determinada (el término *Presidente*(*EEUU*) no puede denotar diferentes objetos dependiendo del valor de *t*, puesto que la ontología utilizada mantiene los índices de tiempo separados de los flujos). La única posibilidad es que *Presidente*(*EEUU*) denote un objeto simple que consista en gente diferente en diferentes momentos en el tiempo. Es el objeto que representa a George Washington desde 1789 hasta 1796, John Adams desde 1976 hasta 1800 y así sucesivamente, como muestra la Figura 10.6.

Figura 10.6 Vista esquemática del objeto *Presidente*(*EEUU*) desde los 15 primeros años de su existencia.

Para expresar que George Washington fue presidente durante 1790, se puede escribir

$$T(Presidente(EEUU) = George\ Washington, 1790DC)$$

Sin embargo, se necesita ser cuidadoso. En la sentencia anterior, el símbolo igual «=» debe ser un símbolo de función en lugar de un operador lógico estándar. La interpreta-

ción *no* es que *George Washington* y *Presidente*(*EEUU*) son idénticos desde un punto de vista lógico en 1790. La identidad lógica no es algo que pueda cambiar a lo largo del tiempo. La identidad lógica existe entre los subeventeos de cada objeto que están definidos por el período 1790.

No confundir el objeto físico *George Washington* con una colección de átomos. *George Washington* no es idéntico desde un punto de vista lógico a *ninguna* colección específica de átomos, porque el conjunto de átomos que lo forman varía considerablemente en el tiempo. Él tiene un tiempo de vida corto y cada átomo tiene un tiempo de vida muy corto. En conjunto, todos se cruzan durante algún período de tiempo, durante el cual, el tiempo de vida del átomo es *ParteDe* George y a partir de ahí evolucionan por separado.

10.4 Eventos mentales y objetos mentales

Los agentes que se han construido hasta ahora tienen creencias y pueden deducir nuevas creencias. Pero por ahora ninguno de ellos tiene conocimiento *sobre* creencias o *sobre* deducción. En dominios de un solo agente simple, el conocimiento sobre el propio conocimiento y los procesos de razonamiento son útiles para controlar la inferencia. Por ejemplo, si uno es consciente de que no puede conocer nada acerca de la geografía rumana, entonces no empleará un enorme esfuerzo computacional para tratar de calcular el camino más corto desde Arad a Bucarest. Uno también puede razonar sobre su propio conocimiento, para construir planes que permitan cambiarlo (por ejemplo comprando un mapa de Rumanía). En dominios multiagente, es importante para un agente razonar acerca de los estados mentales de los otros agentes. Por ejemplo, un oficial de policía rumano posiblemente conocerá la mejor forma de llegar a Bucarest, por lo que el agente podría pedirle ayuda.

En esencia, lo que se precisa es un modelo de los objetos mentales que existen en la cabeza (o en la base de conocimiento) de alguien y los procesos mentales para manipular esos objetos mentales. El modelo debe ser fiel a la realidad, pero no tiene por qué ser detallado. No se necesita ser capaz de predecir cuántos milisegundos le llevará a un agente en concreto realizar una deducción, ni tampoco será necesario predecir qué neuronas se dispararán cuando a un animal se le presenta un determinado estímulo visual. Llegará con concluir que el oficial de policía rumano podría informar de cómo llegar a Bucarest si conoce el camino y se da cuenta de que alguien está perdido.

Una teoría formal de creencias

ACTITUDES DE PROPOSICIÓN

Se comenzó trabajando con las relaciones existentes entre agentes y «objetos mentales» (relaciones como *Cree*, *Conoce* y *Desea*). Las relaciones de este tipo se denominan **actitudes de proposición**, porque describen una actitud que un agente puede tomar hacia una proposición. Supóngase que Lois cree algo, es decir, *Cree*(*Lois, x*). ¿Qué tipo de cosa es *x*? Obviamente *x* no puede ser una sentencia lógica. Si *Vuela*(*Supermán*) es una sentencia lógica, no se puede decir *Cree*(*Lois*, *Vuela*(*Supermán*)), porque sólo los términos (no las sentencias) pueden ser argumentos de los predicados. Pero si *Vuela* es una fun-

ción, entonces *Vuela*(*Supermán*) es un candidato para ser un objeto mental, y *Cree* puede ser una relación entre una gente y un flujo proposicional. Convertir una proposición en un objeto, se conoce con el nombre de **reificación**[6].

REIFICACIÓN

Esto parece proporcionar lo que se necesita: la capacidad para que un agente razone sobre las creencias de los agentes. Desafortunadamente, hay un problema con esta aproximación: si Clark y Supermán son uno al mismo tiempo (es decir, *Clark* = *Supermán*) entonces los vuelos de Clark y Supermán son el mismo y pertenecen a la misma categoría de eventos, es decir, *Vuela*(*Clark*) = *Vuela*(*Supermán*). Por lo tanto, se debe concluir que si Lois cree que Supermán puede volar, él también cree que Clark puede volar, *incluso si ella no cree que Clark es Supermán*. Es decir,

$$(Supermán = Clark) \models (Cree(Lois, Vuela(Supermán)) \Leftrightarrow Cree(Lois, Vuela(Clark)))$$

Una de las interpretaciones de lo anterior es válida: Lois cree que cierta persona, que a veces se llama Clark, puede volar. Pero existe otra interpretación que es errónea: si se le pregunta a Lois ¿puede volar Clark?, indudablemente contestaría que no. Los objetos reificados y los eventos funcionan bien para la primera interpretación de *Cree*, pero para la segunda interpretación sería necesario reificar las *descripciones* de tales objetos y eventos, de manera que Clark y Supermán puedan ser descripciones distintas (aunque se refieran al mismo objeto).

Desde un punto de vista técnico, a la propiedad de sustituir libremente un término por otro igual se le denomina **transparencia referencial**. En lógica de primer orden, todas las relaciones tienen trasparencia referencial. Convendría definir *Cree* (y las otras actitudes de proposición) como relaciones cuyo segundo argumento es **opaco** referencialmente, es decir, no es posible sustituir el segundo argumento por un término igual sin cambiar el significado.

TRANSPARENCIA REFERENCIAL

OPACO

Existen dos formas de lograr esto. La primera es usar una lógica diferente, denominada **lógica modal**, en la cual las actitudes de proposición como *Cree* y *Sabe* son **operadores modales** que son referencialmente opacos. Esta aproximación se trata en la sección de notas históricas. La segunda aproximación que se desea lograr, es conseguir una opacidad efectiva con un lenguaje transparente referencialmente usando una **teoría sintáctica** de objetos mentales. Esto significa que los objetos mentales serán representados por **cadenas**. El resultado es un modelo rudimentario de la base de conocimiento de un agente, que consiste en cadenas que representan sentencias que son creídas por el agente. Una cadena no es más que un término complejo denotado por una lista de símbolos, así, el evento *Vuela*(*Clark*), puede ser representado por la lista de caracteres [*V,u,e,l,a,(,C,l,a,r,k,)*], que se abreviará como «*Vuela*(*Clark*)». La teoría sintáctica incluye un **axioma de cadena única** que establece que las cadenas son idénticas si y sólo si están formadas por caracteres idénticos. Por lo tanto, incluso si *Clark* = *Supermán*, se seguirá teniendo que «*Clark*» ≠ «*Supermán*».

LÓGICA MODAL

OPERADORES MODALES

TEORÍA SINTÁCTICA

CADENAS

AXIOMA DE CADENA ÚNICA

Ahora sólo falta proponer sintaxis, semántica y teoría de la demostración correspondientes al lenguaje de representación de la cadena, como en el caso del Capítulo 7.

---

[6] El término «reificación» viene de la palabra Latina *res*, o cosa. John McCarthy propuso el término «cosificación», pero nunca llegó a ser popular.

La diferencia es que ahora hay que definirlas a todas mediante lógica de primer orden. Se comenzará por definir *Den* como aquella función que correlaciona una cadena con el objeto que denota ésta, y a *Nombrar* como la función que correlaciona un objeto con una cadena que es el nombre de una constante que denota al objeto. Por ejemplo la denotación de «*Clark*» como «*Supermán*» es el objeto referido mediante el símbolo constante *HombreDeAcero*, y el nombre de tal objeto puede ser tanto «*Supermán*» como «*Clark*», o alguna otra constante, por ejemplo, «X_{11}».

$$Den(«Clark») = HombreDeAcero \wedge Den(«Supermán») = HombreDeAcero$$
$$Nombrar(HombreDeAcero) = «X_{11}»$$

El siguiente paso consiste en definir las reglas de inferencia de los agentes lógicos. Por ejemplo, digamos que se desea que el agente lógico sea capaz de efectuar un *Modus Ponens*: si cree p y también cree $p \Rightarrow q$, entonces también cree q. El primer ensayo de cómo escribir este axioma es:

$$AgenteLógico(a) \wedge Cree(a, p) \wedge Cree(a, «p \Rightarrow q») \Rightarrow Cree(a, q)$$

Pero es incorrecto, puesto que aunque la cadena «$p \Rightarrow q$» contiene las letras «p» y «q», no tiene nada que ver con las cadenas que corresponden a los valores de las variables p y q. La formulación correcta es

$$AgenteLógico(a) \wedge Cree(a, p) \wedge Cree(a, Concatenar(p, «\Rightarrow», q)) \Rightarrow Cree(a, q)$$

donde *Concatenar* es una función de las cadenas que concatena entre sí a sus elementos. Se abreviará $Concatenar(p, «\Rightarrow», q)$ como «$\underline{p} \Rightarrow \underline{q}$». Es decir, la aparición de $\underline{x}$ dentro de una cadena es **no acotada**, con el significado de que se debe sustituir la variable x por su valor. Los programadores de Lisp identificarán lo anterior como un operador de cita reversiva, y los programadores de Perl lo reconocerán como una interpolación de una variable de tipo $.

NO ACOTADA

Una vez añadidas las otras reglas de inferencia además de Modus Ponens, se podrá responder a preguntas como «si un agente lógico conoce estas premisas, ¿podrá llegar a esa conclusión?». Además de las reglas de inferencia normales, también son necesarias algunas reglas específicas de creencia. Por ejemplo, la siguiente regla afirma que si un agente lógico cree algo, entonces también cree que lo cree:

$$AgenteLógico(a) \wedge Cree(a, p) \Rightarrow Cree(a, «Cree(\underline{Nombrar(a)}, \underline{p})»)$$

A partir de ahora, de acuerdo con el axioma, un agente puede deducir cualquier consecuencia de su creencia de un modo infalible. Esto se llama **omnisciencia lógica**. Se han realizado numerosos intentos para tratar de definir agentes racionales limitados, los cuales pueden realizar un número limitado de deducciones en un tiempo también limitado. Nada es completamente satisfactorio, pero esta formulación permite un rango muy restringido de predicciones sobre agentes limitados.

OMNISCIENCIA LÓGICA

Conocimiento y creencia

Las relaciones entre creencia y conocimiento se han estudiado de forma exhaustiva en Filosofía. Es conocida la afirmación en el sentido de que el conocimiento no es sino una creencia válida demostrada. Por ejemplo, si usted cree algo, y si esto último realmente

es cierto, y tiene la prueba de ello, entonces usted lo sabe. La demostración es indispensable, pues evitará que usted afirme «Sé que el resultado de lanzar esta monda al aire será una cara» y que sólo sea cierto la mitad de las veces.

Considérese que *Sabe*(*a, p*) significa que el agente *a sabe que* la preposición *p* es cierta. También es posible definir otras clases de conocimiento. Por ejemplo, a continuación se establece una definición de «saber si»:

$$SabeSi(a, p) \Leftrightarrow Sabe(a, p) \vee Sabe(a, «\neg \underline{p}»)$$

Continuando con el ejemplo, Lois sabe si Clark puede volar tanto si ella sabe que Clark puede volar, como si no.

El concepto de «saber que» es más complicado. Uno puede sentirse tentado a aceptar que un agente sabe que el número de teléfono de Bob es una *x* para la cual *x* = *NúmeroTeléfono*(*Bob*). Pero esto no es suficiente, porque el agente podría conocer que Alice y Bob tienen el mismo número de teléfono (es decir, *NúmeroTeléfono*(*Bob*) = *NúmeroTeléfono*(*Alice*)), pero si el número de teléfono de Alice es desconocido, esto no sería de mucha ayuda. Una definición mejor de «saber que» sería que el agente debe saber cierta *x* que es una cadena de dígitos y que es el número de Bob:

$$SabeQue(a, «NúmeroTeléfono(\underline{b})») \Leftrightarrow$$
$$\exists\, x \;\; Sabe(a, «\underline{x} = NúmeroTeléfono(\underline{b})») \wedge x \in SecuenciaDeDígitos$$

Desde luego que para otro tipo de preguntas los criterios serán disjuntos, dependiendo de lo que se considere como una respuesta aceptable. En el caso de la pregunta «¿cuál es la capital de Nueva York?», una respuesta aceptable sería un nombre propio, «Albani», no algo como «la ciudad en donde está la cámara legislativa del estado». Para ello, se hará que *SaberQue* sea una relación de tres partes: empleará un agente, un término y un predicado que integren la respuesta válida. Por ejemplo, se podría tener lo siguiente:

$$SabeQue(Agente, «Capital(NuevaYork)», NombrePropio)$$
$$SabeQue(Agente, «NúmeroTeléfono(Bob)», SecuenciaDeDígitos)$$

Conocimiento, tiempo y acción

En la mayoría de las situaciones reales, un agente tratará con creencias (las suyas propias o las de otros agentes) que cambian en el tiempo. El agente también tendrá que hacer planes que involucren cambios de sus propias creencias, como comprar un mapa para averiguar cómo llegar a Bucarest. Como con otros predicados, se podrá reificar *Cree* y hablar sobre creencias que ocurren en un período de tiempo. Por ejemplo, para decir que Lois cree hoy que Supermán puede volar, se escribirá

$$T(Cree(Lois, «Volar(Supermán)»), Hoy)$$

Si el objeto de creencia es una proposición que puede cambiar en el tiempo, entonces también se puede describir utilizando el operador *T* dentro de la cadena. Por ejemplo, Lois podría creer hoy que Supermán pudo volar ayer:

$$T(Cree(Lois, «T(Volar(Supermán), Ayer)», Hoy)$$

PRECONDICIONES DE CONOCIMIENTO

EFECTOS DE CONOCIMIENTO

Dada una forma de describir las creencias en el tiempo, se puede usar la maquinaría del cálculo de eventos para realizar planes que incluyan creencias. Acciones que pueden tener **precondiciones de conocimiento** y **efectos de conocimiento**. Por ejemplo, la acción de marcar un número de teléfono tiene la precondición de conocer dicho número, y la acción de buscar el número tiene el efecto de conocer el número. Se puede describir esta última acción a través de la maquinaria del cálculo de eventos:

$$Iniciar(Buscar(a, «NúmeroTeléfono(\underline{b})»),$$
$$SabeQue(a, «NúmeroTeléfono(\underline{b})», SecuenciaDeDígitos), t)$$

VARIABLES DE EJECUCIÓN

Las **variables de ejecución** se utilizan frecuentemente como notación abreviada para representar planes que unen y utilizan información. Esto es similar a la convención de variable no acotada vista con anterioridad. Por ejemplo, el plan para buscar el número de teléfono de Bob y llamarlo a continuación puede ser escrito como

$$[Buscar(Agente, «NúmeroTeléfono(Bob)», \underline{n}), Llamar(\underline{n})]$$

Aquí, $\underline{n}$ es una variable de ejecución cuyo valor será establecido por la acción *Buscar* y puede ser usado por la acción *Llamar*. Los planes de este tipo ocurren frecuentemente en dominios parcialmente observables. Se verán ejemplos en la sección siguiente y en el Capítulo 12.

10.5 El mundo de la compra por Internet

En esta sección se codificará algún conocimiento relacionado con la compra a través de Internet. Se creará un agente que investigue compras y ayude al comprador a encontrar ofertas de productos en Internet. El comprador da al agente de compra una descripción de un producto, y el agente debe producir un listado de páginas web que oferten ese producto. En algunos casos, la descripción del producto que da el comprador será precisa, como para la *cámara digital Coolpix 995*, y la tarea será encontrar aquellas tiendas que tengan una mejor oferta. En otros casos la descripción será sólo parcialmente especificada, como *una cámara digital por un precio inferior a 300 dólares,* y el agente tendrá que comparar productos diferentes.

El entorno del agente de compra es toda la WWW (*Word Wide Web*), no un entorno de juguete simulado, sino un entorno complejo que evoluciona constantemente y que es usado por millones de personas cada día. La percepción del agente serán las páginas web, pero mientras un usuario de la Web percibe las páginas como vector de puntos en pantalla, el agente de compra recibirá la página como una cadena de caracteres, que consiste en un conjunto de palabras ordinarias intercaladas con comandos de formato en lenguaje HTML. La Figura 10.7 muestra una página web y su cadena de caracteres HTML correspondiente. El problema de la percepción para el agente de compra abarca la extracción de la información útil de percepciones de este tipo.

Claramente, la percepción de páginas web es más fácil que, por ejemplo, la percepción mientras se conduce un taxi en El Cairo. Sin embargo, existen complicaciones en la tarea de percepción en Internet. La página web de la Figura 10.7 es muy simple com-

Almacén Genérico online

Seleccionar una de nuestras trabajadas líneas de productos:

- Computadores
- Cámaras
- Libros
- Vídeos
- Música

```
<h1>Almacén Genérico on-line</h1>
<i>Seleccionar</i> una de nuestras trabajadas líneas de productos:
<ul>
<li> <a href=«http://gen-store.com/compu»>Computadores</a>
<li> <a href=«http://gen-store.com/cama»>Cámaras</a>
<li> <a href=«http://gen-store.com/libr»>Libros</a>
<li> <a href=«http://gen-store.com/vide»>Vídeos</a>
<li> <a href=«http://gen-store.com/musi»>Música</a>
</ul>
```

Figura 10.7 Una página web procedente de un almacén genérico *online* en la forma percibida por una persona (arriba), y la correspondiente cadena HTML percibida por el navegador web o el agente de compra (abajo). En HTML, los caracteres entre < y > son etiquetas que especifican cómo se muestra la página. Por ejemplo, la cadena `<i>Seleccionar</i>` implica establecer una fuente en cursiva, mostrar la palabra *Seleccionar* y finalizar con el uso de la fuente cursiva. Un identificador de página como `http://gen-store.com/libr` se denomina un **localizador de recursos uniforme** o URL. La etiqueta `<a href=`"*url*"`>`*ancla*`</a>` implica la creación de un hipervínculo a la *url* con el **texto del enlace** *ancla*.

parada con webs reales dedicadas a la compra de artículos, que incluyen cookies, Java, Javascript, Flash, protocolos de exclusión de robots, código HTML con incorrecciones, ficheros de sonido, clips de películas y texto que aparece sólo como imágenes JPEG. Un agente que sea capaz de tratar con *todo* Internet es casi tan complejo como un robot que pueda mover un objeto en el mundo real. El planteamiento se centrará en un agente que ignore la mayoría de estas complicaciones.

La primera tarea del agente es encontrar ofertas relevantes de productos (se verá posteriormente cómo seleccionar la mejor oferta de las relevantes). Sea *petición* la descripción del producto que el usuario teclea (por ejemplo, «portátiles»). Entonces una página es una oferta relevante para *petición* si la página es relevante y la página es efectivamente una oferta. También se seguirá la pista de la URL asociada con la página:

$$OfertaRelevante(página, url, petición) \Leftrightarrow Relevante(página, url, petición) \wedge Oferta(página)$$

Una página con una comparativa de los portátiles más modernos debería ser relevante, pero si no proporciona un mecanismo para comprar, no es una oferta. Por ahora, se dirá que

una página es una oferta si contiene la palabra «compra» o «precio» en un enlace HTML o en un formulario. En otras palabras, si la página contiene una cadena de la forma: «`<a ...compra ..</a`» entonces es una oferta. También podría aparecer «`precio`» en vez de «`compra`» o usar la etiqueta «`form`» en vez de la etiqueta «`a`». Se puede escribir un axioma para esto:

$$
\begin{array}{lll}
Oferta(página) & \Leftrightarrow & EnEtiqueta(«a», str, página) \vee \\
 & & \quad EnEtiqueta(«form», str, página)) \\
 & & \quad \wedge (En(«comprar», str) \\
 & & \quad\quad \vee (En(«precio», str)). \\
EnEtiqueta(etiqueta, cadena, página) & \Leftrightarrow & En(« < » + etiqueta + str + \\
 & & \quad « < /» + etiqueta, página). \\
En(subcadena, cadena) & \Leftrightarrow & \exists\, i\ str[i : i + Longitud(subcadena)] \\
 & & \quad = subcadena.
\end{array}
$$

Ahora se necesita encontrar páginas relevantes. La estrategia es empezar en la página de inicio de un almacén general y considerar todas las páginas a las que se puede llegar siguiendo los enlaces relevantes[7]. El agente tendrá conocimiento acerca de varios almacenes, por ejemplo:

$$
\begin{array}{l}
Amazon \in AlmacenesOnLine \wedge PáginaDeInicio(Amazon, «amazon.com»). \\
Ebay \in AlmacenesOnLine \wedge PáginaDeInicio(Ebay, «ebay.com»). \\
GenStore \in AlmacenesOnLine \wedge PáginaDeInicio(GenStore, «gen\text{-}store.com»)
\end{array}
$$

Los almacenes clasifican sus productos en categorías de productos, y proporcionan enlaces a las categorías principales desde sus páginas de inicio. Las categorías descendientes se pueden localizar siguiendo los enlaces relevantes, y finalmente se accederá a las ofertas. En otras palabras, una página es relevante para la petición si se puede localizar a través de enlaces relevantes de categorías que parten de la página principal del almacén, y entonces se seguirán uno o más enlaces a ofertas de productos:

$$
\begin{array}{l}
Relevante(página, url, petición) \Leftrightarrow \\
\quad \exists\, almacén, inicio\ \ almacén \in AlmacenesOnLine \wedge \\
\quad\quad PáginaDeInicio(almacén, inicio) \\
\quad\quad\quad \wedge \exists\, url_2\ \ EnlacesRelevante(inicio, url_2, petición) \wedge Enlace(url_2, url) \\
\quad\quad\quad\quad \wedge\, página = ObtenerPágina(url)
\end{array}
$$

Aquí el predicado *Enlace*(*de*, *a*) significa que existe un hipervínculo desde la URL *de* hasta la URL *a* (*véase* Ejercicio 10.13). Para definir qué se tiene en cuenta en *EnlacesRelevante*, no se podrá seguir cualquier hipervínculo, sino aquellos cuyo texto indique que es relevante para la petición del producto. Para ello, se utilizarán *TextoDeEnlace*(*de, a, texto*) para significar que hay un enlace entre *de* hacia *a* con el texto especificado por *texto*. Una cadena de enlaces entre dos URL's, *inicio* y *fin*, es relevante para una descripción *d* si el texto del hipervínculo de cada enlace contiene un nombre de categoría

[7] Una alternativa a la estrategia de seguir los enlaces es la de usar los resultados de un motor de búsqueda de Internet. La tecnología que hay detrás de un buscador de Internet, recuperación de información, será tratada en la Sección 23.2.

relevante para *d*. La existencia de la cadena de enlaces en sí misma se determina por una definición recursiva con el enlace vacío (*inicio* = *fin*) como caso base:

$$\begin{aligned} &EnlacesRelevantes(inicio, fin, petición) \Leftrightarrow (inicio = fin) \\ &\quad \vee (\exists\, u, texto\ \ TextoDeEnlace(inicio, u, texto) \\ &\qquad \wedge NombreCategoríaRelevante(petición, texto) \\ &\qquad\quad \wedge EnlacesRelevante(u, fin, petición)) \end{aligned}$$

Ahora se puede definir lo que debe contener *texto* para que sea un *NombreCategoríaRelevante* para *petición*. Esto se realiza utilizando el predicado *Nombre*(*s*, *c*), que dice que la cadena *s* es un nombre de categoría *c* (por ejemplo, se puede afirmar que *Nombre*(«*portátiles*», *ComputadoresPortátiles*). Algunos ejemplos más del predicado *Nombre* aparecen en la Figura 10.8(b). A continuación, se definirá el concepto de relevancia. Supóngase que la *petición* es «portátiles». Entonces *NombreCategoríaRelevante*(*petición*, *texto*) es cierto cuando una de las siguientes afirmaciones es cierta:

- El *texto* y la *petición* hacen referencia a la misma categoría (por ejemplo, «computadores portátiles» y «portátiles»).
- El *texto* hace referencia a una supercategoría como «computadores».
- El *texto* hace referencia a una subcategoría como «*notebooks* ultraligeros».

La definición lógica de *NombreCategoríaRelevante* es la siguiente:

$$\begin{aligned} &NombreCategoríaRelevante(petición, texto) \Leftrightarrow \\ &\quad \exists\, c_1, c_2\ \ Nombre(petición, c_1) \wedge Nombre(texto, c_2) \wedge (c_1 \subseteq c_2 \vee c_2 \subseteq c_1) \end{aligned} \tag{10.1}$$

Aparte de esto, el texto del hipervínculo es irrelevante porque se refiere a una categoría fuera de esta línea, como «computadores *mainframe*» o «césped & jardín».

Para seguir enlaces relevantes, es esencial tener una jerarquía rica en categorías de productos. La parte superior de esta jerarquía debería ser parecida a la mostrada en la Figura 10.8(a). No será viable realizar un listado de todas las categorías de compras posibles, porque a un comprador le podrían surgir nuevos deseos y los fabricantes crearían nuevos productos para satisfacer a los clientes (¿calentadores de rótula eléctricos?). Sin

(a)	(b)
Libros ⊂ *Productos*	*Nombre*(«*libros*», *Libros*)
GrabacionesMusicales ⊂ *Productos*	*Nombre*(«*música*», *GrabacionesMusicales*)
CDsMusicales ⊂ *GrabacionesMusicales*	*Nombre*(«*CDs*», *CDsMusicales*)
CintasMúsica ⊂ *GrabacionesMusicales*	*Nombre*(«*cintas*», *CintasMúsica*)
Electrónica ⊂ *Productos*	*Nombre*(«*electrónica*», *Electrónica*)
CámarasDigitales ⊂ *Electrónica*	*Nombre*(«*cámaras digitales*», *CámarasDigitales*)
EquipamientoEstéreo ⊂ *Electrónica*	*Nombre*(«*estéreos*», *EquipamientoEstéreo*)
Computadores ⊂ *Electrónica*	*Nombre*(«*computadores*», *Computadores*)
ComputadoresPortátiles ⊂ *Computadores*	*Nombre*(«*portátiles*», *ComputadoresPortátiles*)
ComputadoresSobreMesa ⊂ *Computadores*	*Nombre*(«*sobremesa*», *ComputadoresSobreMesa*)
...	...

Figura 10.8 (a) Taxonomía de categorías de productos. (b) Palabras usadas para esas categorías.

embargo, una ontología de aproximadamente 1.000 categorías sería una herramienta muy útil para la mayoría de los compradores.

Además de la jerarquía de productos en sí misma, también es necesario tener un vocabulario rico para los nombres de las categorías. La vida sería mucho más sencilla si hubiera una correspondencia uno a uno entre categorías y cadenas de caracteres para nombrarlas. Ya se ha comentado el problema de la **sinonimia** (dos nombres para la misma categoría, como «computadores portátiles» y «portátiles»). También existe el problema de la **ambigüedad** (un nombre para dos o más categorías diferentes). Por ejemplo, si se añade la sentencia

Nombre(*«CDs»*, *CertificadosDeDepósito*)

a la base de conocimiento de la Figura 10.8(b), entonces «CDs» nombrará a dos categorías diferentes.

Sinonimia y ambigüedad pueden causar un incremento significativo en el número de caminos que el agente debe seguir, y puede algunas veces hacer difícil el determinar si una página dada es realmente relevante. Un problema mucho más serio es la existencia de un amplio rango de descripciones que un usuario puede teclear, así como nombres de categorías que un almacén puede emplear. Por ejemplo, un enlace podría decir «portátil» cuando la base de conocimiento tiene sólo «portátiles», o el usuario podría preguntar por «un computador que pueda encajar en una mesa portátil de un asiento de clase económica en un Boeing 737». Resulta imposible enumerar a priori todas las formas de nombrar a una categoría, por lo que el agente tendrá que ser capaz de realizar un razonamiento adicional para determinar si la relación *Nombre* es cierta. En el peor caso, esto requiere una comprensión completa del lenguaje natural, un tema que se postergará hasta el Capítulo 22. En la práctica, unas pocas reglas simples (como el permitir que «portátil» case con una categoría llamada «portátiles») sigue un largo camino. El Ejercicio 10.15 propone el desarrollo de un conjunto de reglas después de hacer algunas indagaciones en los almacenes *online*.

Dadas las definiciones lógicas de los párrafos precedentes y las bases de conocimiento apropiadas sobre categorías de producto y convenciones de nombres, ¿se está preparado para aplicar un algoritmo de inferencia y obtener un conjunto de ofertas relevantes para la petición?, ¡no exactamente! El elemento que falta es la función *ObtenerPágina*(*url*), que se refiere a la página HTML de una URL determinada. El agente no tiene el contenido de cada URL en su base de conocimiento, ni tampoco reglas explícitas para deducir qué tipo de contenidos encontrará. En su lugar, se puede hacer que el procedimiento HTTP correcto se ejecute siempre que un subobjetivo necesite la función *ObtenerPágina*. De esta forma, se hace creer al motor de inferencia que la Web entera se encuentra dentro de la base de conocimiento. Esto es un ejemplo de la técnica general denominada **acoplamiento funcional**, según el cual predicados y funciones particulares pueden ser manejadas por métodos de propósito específico.

ACOPLAMIENTO FUNCIONAL

Comparación de ofertas

Asúmase que el proceso de razonamiento de la sección precedente ha producido un conjunto de páginas con ofertas para la petición sobre «portátiles». Para comparar esas ofer-

tas, el agente debe extraer la información relevante (precio, velocidad, tamaño del disco duro, peso, etc.) de las páginas recuperadas. Esto puede ser una tarea difícil en páginas web reales por las razones que se comentaron con anterioridad. Un modo común de abordar este problema es usar algunos programas de tipo **envoltorio** para extraer información de la página. La tecnología para la extracción de información se trata en la Sección 23.3. Por ahora, se asumirá que los programas de tipo envoltorio existen, y cuando se les proporciona una página determinada y una base de conocimiento, añaden aserciones a la base de conocimiento. Normalmente se aplicará una jerarquía de programas de tipo envoltorio a una página: uno muy general para extraer fechas y precios, uno más específico para extraer atributos de productos relacionados con los computadores y, si es necesario, uno específico para el sitio web particular y que conozca el formato concreto del almacén. Dada una página en el sitio de gen-store.com con el texto

ENVOLTORIO

```
YVM ThinkBook 970.     Nuestro Precio: 1449.00$
```

seguido de varias especificaciones técnicas, se deseará un programa envoltorio para extraer información como la siguiente:

$$\begin{aligned}&\exists\, lc, oferta \;\; lc \in ComputadoresPortátiles \wedge oferta \in OfertaDeProductos \wedge \\ &TamañoPantalla(lc, Pulgadas(14)) \wedge TipoPantalla(lc, ColorLCD) \wedge \\ &TamañoMemoria(lc, Megabytes(512)) \wedge VelocidadCPU(lc, GHz(2.4)) \wedge \\ &ProductoOfrecido(oferta, lc) \wedge Almacén(oferta, GenStore) \wedge \\ &URL(oferta, \text{«genstore.com/comps/34356.}html\text{»}) \wedge \\ &Precio(oferta, \$(449)) \wedge Fecha(oferta, Hoy)\end{aligned}$$

Este ejemplo ilustra varios temas que surgen cuando se toma en serio la tarea de aplicar ingeniería de conocimiento a las transacciones comerciales. Por ejemplo, destacar que el precio es un atributo de una *oferta*, no del producto en sí mismo. Esto es importante porque la oferta en un almacén determinado puede cambiar día a día, incluso para el mismo portátil. Para algunas categorías (como casas y pinturas) el mismo objeto individual puede ser ofrecido simultáneamente por diferentes intermediarios a diferentes precios. Todavía hay más complicaciones que aún no se han manejado, como la posibilidad de que el precio dependa del método de pago y de la clasificación del comprador para ciertos descuentos. Sea como sea, todavía queda un montón de trabajo interesante por hacer.

La tarea final es comparar las ofertas que han sido extraídas. Por ejemplo, considérense estas tres ofertas:

A : 2.4 GHz CPU, 512MB RAM, 80 GB disk, DVD, CDRW, 1695$.
B : 2.0 GHz CPU, 1GB RAM, 120 GB disk, DVD, CDRW, 1800$.
C : 2.2 GHz CPU, 512MB RAM, 80 GB disk, DVD, CDRW, 1800$.

C es **dominada** por *A*, es decir, *A* es más barato y más rápido, y todas las demás características son iguales. En general, *X* domina a *Y* si *X* tiene un valor mejor en al menos un atributo, y no es peor en los atributos restantes. Pero ni *A* ni *B* dominan la una a la otra. Para decidir cuál es mejor, se necesita conocer cómo sopesa el comprador la velocidad de CPU y el precio, frente a la memoria y la capacidad en disco. El tema general de preferencias entre múltiples atributos se aborda en la Sección 16.4. Por ahora, el agen-

te de compra simplemente devolverá una lista con todas las ofertas no dominantes que cumplen con la descripción del comprador. En este ejemplo, tanto *A* como *B* no son dominantes. Nótese que esta salida se basa en la asunción de que todo el mundo prefiere precios más baratos, procesadores más rápidos y más capacidad de almacenamiento. Algunos atributos, como el tamaño de pantalla en un portátil, dependen de las preferencias particulares del usuario (portabilidad vs. visibilidad). Para éstas, el agente de compra preguntará al usuario.

El agente de compra que se ha descrito aquí es muy simple y se pueden realizar muchos refinamientos. De todos modos, tiene suficiente capacidad para ser utilizado como asistente para la compra, siempre y cuando se disponga de conocimiento correcto acerca del domino específico. Debido a su construcción declarativa, es fácilmente extensible para aplicaciones más complejas. El punto principal en esta sección es mostrar que es necesaria una representación del conocimiento (en particular de la jerarquía de productos), para la construcción de un agente de este tipo, y que una vez que se dispone de conocimiento en esta forma, no es muy complicado diseñar un agente basado en conocimiento que haga el resto.

10.6 Sistemas de razonamiento para categorías

Se han visto las categorías como bloques de construcción primarios para cualquier esquema de representación del conocimiento a gran escala. Existen dos familias de sistemas íntimamente relacionadas: (i) las **redes semánticas** proporcionan una ayuda gráfica para visualizar una base de conocimiento, así como algoritmos eficientes para inferir propiedades de un objeto con base en su pertenencia a una categoría; (ii) la **lógica descriptiva** proporciona un lenguaje formal para construir y combinar definiciones de categorías, así como algoritmos eficientes para decidir las relaciones de subconjunto y superconjunto entre categorías.

Redes semánticas

GRAFOS EXISTENCIALES

En 1909, Charles Peirce propuso una notación gráfica de nodos y arcos denominada **grafos existenciales** que él denominó «la lógica del futuro». Entonces empezó un debate de larga duración entre los defensores de la «lógica» y los defensores de las «redes semánticas». Desafortunadamente, el debate oscureció el hecho de que las redes semánticas (por lo menos aquellas con un concepto de semántica bien definido) son una forma de lógica. La notación que proporcionan las redes semánticas para cierta clase de sentencias es a menudo más conveniente, pero si se deja de lado la «interfaz humana», los conceptos base (objetos, relaciones, cuantificación, etc.) son los mismos.

Existen diversas variantes de las redes semánticas, pero todas son capaces de representar objetos individuales, categorías de objetos y relaciones entre objetos. Una notación gráfica común visualiza objetos o nombres de categorías en óvalos o cajas, y los conecta con arcos etiquetados. Por ejemplo, la Figura 10.9 tiene un enlace *MiembroDe* entre *Mary* y *PersonaFemenina*, correspondiente a la aserción lógica $Mary \in Persona$

Figura 10.9 Predicados sobre intervalos de tiempo.

Femenina. De igual modo, el enlace *HemanaDe* entre *Mary* y *John* corresponde a la aserción *HermanaDe*(*Mary, John*). Se pueden conectar categorías usando enlaces *SubconjuntoDe*. Es tan divertido dibujar burbujas y flechas que uno puede dejarse llevar. Por ejemplo, se conoce que todas las personas tienen una madre de sexo femenino, por lo tanto ¿se puede dibujar un enlace *TieneMadre* desde *Personas* hasta *PersonaFemenina*? La respuesta es no, porque *TieneMadre* es una relación entre una persona y su madre, y las categorías no tienen madre[8]. Por esta razón, se ha usado una notación especial (en enlace con caja doble) en la Figura 10.9. Este enlace expresa que

$$\forall\, x \;\; x \in Personas \Rightarrow [\forall\, y \;\; TieneMadre(x, y) \Rightarrow y \in PersonaFemenina]$$

También se podría querer afirmar que las personas tienen dos piernas, por lo que,

$$\forall\, x \;\; x \in Personas \Rightarrow Piernas(x, 2)$$

Como antes, se necesita ser cuidadoso para no afirmar que una categoría tiene piernas. La caja con línea simple de la Figura 10.9 se usa para especificar propiedades de cada miembro de una categoría.

La notación de la red semántica hace muy conveniente la utilización de razonamiento basado en **herencia** del tipo del introducido en la Sección 10.2. Por ejemplo, por el hecho de ser persona, Mary hereda la propiedad de tener dos piernas. Por lo tanto, para saber cuántas piernas tiene Mary, el algoritmo de herencia sigue el enlace *MiembroDe* desde *Mary* hasta la categoría a la cual pertenece y entonces continúa por el enlace *SubconjuntoDe* hasta encontrar la categoría en la cual existe un enlace etiquetado con el recuadro *Piernas* (en este caso, la categoría *Personas*). La simplicidad y eficiencia de este mecanismo de inferencia, comparado con el teorema de la prueba lógica, ha sido uno de los más atractivos para las redes semánticas.

[8] Varios sistemas desarrollados hace tiempo fallaron en distinguir entre propiedades de miembros de una categoría y propiedades de la categoría como un todo. Esto puede llevar directamente a inconsistencias, como apuntó Drew McDermott (1976) en su artículo *Artificial Intelligence Meets Natural Stupidity*. Otro problema común fue el uso del enlace *EsUn* para los subconjuntos y las relaciones de pertenencia, en correspondencia con el uso inglés: «un gato es un mamífero» y «Fifi es un gato». *Véase* el Ejercicio 10.25 para más información sobre este punto.

HERENCIA MÚLTIPLE

La herencia se complica cuando un objeto puede pertenecer a más de una categoría, o cuando una categoría puede ser un subconjunto de varias categorías. Esto se conoce con el nombre de **herencia múltiple**. En estos casos, el algoritmo de herencia puede encontrar dos o más valores en conflicto que resuelvan la pregunta. Por esta razón, no se permite herencia múltiple en algunos lenguajes de **programación orientada a objetos** (POO), como Java, que usa la herencia para la jerarquía de clases. La herencia múltiple es permitida comúnmente en las redes semánticas, pero se postergará esta discusión hasta la Sección 10.7.

ENLACES INVERSOS

Otra forma común de herencia es el uso de **enlaces inversos**. Por ejemplo, *TieneHermana* es el inverso de *HermanaDe*, lo que significa que

$$\forall\, p, s \;\; TieneHermana(p, s) \;\Leftrightarrow\; HermanaDe(s, p)$$

Esta sentencia puede ser expresada en una red semántica si el enlace es **reificado** (es decir, transformado en objetos). Por ejemplo, se podría tener el objeto *HermanaDe*, conectado con un enlace *inverso* mediante *TieneHermana*. Dada la pregunta acerca de quién es *HermanaDe* John, el algoritmo de inferencia puede descubrir que *TieneHermana* es el inverso de *HermanaDe* y por lo tanto, puede responder a la pregunta siguiendo el enlace *TieneHermana* desde *John* hasta *Mary*. Sin la información inversa, sería necesario comprobar cada persona de sexo femenino para ver si esa persona tiene un enlace *HermanaDe* hacia John. Esto es así debido a que las redes semánticas proporcionan indexación directa sólo para objetos, categorías y los enlaces que salen de ellos. Utilizando el vocabulario de la lógica de primer orden, es como si la base de conocimiento estuviera indexada sólo por el primer argumento de cada predicado.

El lector puede haberse dado cuenta de un inconveniente obvio de la notación de las redes semánticas, en comparación con la lógica de primer orden: el hecho de que los enlaces entre burbujas representan sólo relaciones *binarias*. Por ejemplo, la sentencia *Vuela*(*Shankar, NuevaYork, NuevaDelhi, Ayer*) no se puede expresar directamente en una red semántica. Sin embargo, se *puede* conseguir el efecto de las relaciones *n*-arias reificando la proposición como si fuera un evento (*véase* la Sección 10.3) perteneciente a una categoría de eventos apropiada. La Figura 10.10 muestra la estructura de la red semántica para este evento particular. Nótese que la restricción de las relaciones binarias fuerza a la creación de una ontología rica de conceptos reificados. De hecho, mucha de la ontología desarrollada en este capítulo se originó en sistemas de redes semánticas.

Figura 10.10 Una fragmento de una red semántica que muestra la representación de la aserción lógica *Vuela*(*Shankar*, *NuevaYork*, *NuevaDelhi*, *Ayer*).

La reificación de proposiciones hace posible representar cualquier terreno, sentencia atómica con función libre expresada en lógica de primer orden, utilizando la notación de las redes semánticas. Ciertas clases de sentencias universalmente cuantificadas se pueden expresar usando enlaces inversos y las fechas con cajas de borde simple o doble aplicadas a las categorías, pero esto aún deja un largo camino plagado de lógica de primer orden. La negación, disyunción, símbolos de función anidados y cuantificación existencial aún no se pueden representar. Ahora es *posible* extender la notación para hacerla equivalente a la lógica de primer orden (como en los grafos existenciales de Peirce o las redes semánticas particionadas de Hendrix (1975)), pero haciendo esto se pierde una de las principales ventajas de las redes semánticas, como es la simplicidad y la transparencia del proceso de inferencia. Los diseñadores pueden construir una red grande y seguir teniendo una buena idea sobre qué preguntas serán eficientes, porque (a) es fácil visualizar los pasos que el procedimento de inferencia seguirá y (b) en algunos casos, el lenguaje de consulta es tan simple que las consultas difíciles no se pueden representar. En aquellos casos donde el poder expresivo sea demasiado limitado, algunas redes semánticas proporcionan **acoplamiento procedimental** para rellenar el vacío. El acoplamiento procedimental es una técnica por la cual una consulta sobre (o a veces una afirmación de) una cierta relación, implica una llamada a un procedimiento especial diseñado para esa relación, en lugar de la utilización del algoritmo de inferencia general.

VALORES POR DEFECTO

Uno de los aspectos más importantes de las redes semánticas es su habilidad para representar **valores por defecto** para las categorías. Examinando la Figura 10.9 con detenimiento, uno se da cuenta de que John tiene una pierna en lugar de tener dos, puesto que él es una persona y todas las personas tienen dos piernas. En una base de conocimiento estrictamente lógica esto sería una contradicción, pero en una red semántica la afirmación de que todas las personas tienen dos piernas tiene sólo el sentido de valor por defecto. Es decir, se asume que una persona tiene dos piernas a menos que sea contradicho por información más específica. La semántica por defecto se refuerza de forma natural por el algoritmo de inferencia, porque utiliza enlaces hacia arriba desde el objeto en sí mismo (John en este caso), y para tan pronto como encuentra un valor. Se dice que el valor por defecto es **sobrescrito** por un valor más específico. Nótese que se podría haber sobrescrito el valor por defecto para el número de piernas a través de la creación de una categoría de *PersonasConUnaPierna*, un subconjunto de *Personas* de la cual John es miembro.

SOBREESCRITURA

Se puede mantener la semántica lógica de modo estricto en la red si se dice que la aserción *Personas* posee una excepción para John:

$$\forall\, x \;\; x \in Personas \wedge x \neq John \Rightarrow Piernas(x, 2)$$

Para una red *constante*, lo anterior es semánticamente adecuado, pero será mucho menos concisa que la notación de la red si existen un montón de excepciones. Sin embargo, esta aproximación falla para una red que será actualizada con más aserciones (realmente se quiere decir que cualquier persona con una sola pierna es también una excepción). La Sección 10.7 profundiza en este punto y en el razonamiento por defecto general.

Lógica descriptiva

LÓGICA DESCRIPTIVA

La sintaxis de la lógica de primer orden está diseñada para hacer más fácil el afirmar cosas sobre objetos. La **lógica descriptiva** se basa en notaciones que están diseñadas para hacer más fácil describir definiciones y propiedades de categorías. Los sistemas de lógica descriptiva han evolucionado desde las redes semánticas, en respuesta a las presiones para formalizar el significado de la red y retener al mismo tiempo, el énfasis en la estructura taxonómica como principio organizacional.

SUBSUNCIÓN

CLASIFICACIÓN

Las principales tareas de inferencia para la lógica descriptiva son la **subsunción** (comprobar si una categoría es un subconjunto de otra a través de la comparación de sus definiciones) y la **clasificación** (comprobar si un objeto pertenece a una categoría). Algunos sistemas también incluyen **consistencia** de la definición de una categoría (si el criterio de pertenencia puede ser satisfecho lógicamente).

El lenguaje CLASSIC (Borgida *et al.*, 1989) es un ejemplo típico de lógica descriptiva. En la Figura 10.11 se muestra la sintaxis de las descripciones de CLASSIC[9]. Por ejemplo, para expresar que los solteros son adultos que no están casados, se escribiría

$$Soltero = Y(NoCasado, Adulto, Masculino)$$

El equivalente de lo anterior en lógica de primer orden sería

$$Soltero(x) \Leftrightarrow NoCasado(x) \wedge Adulto(x) \wedge Masculino(x)$$

Obsérvese que la lógica descriptiva realmente permite efectuar operaciones lógicas directas en los predicados, en vez de tener que crear primero oraciones que se unen mediante conectores. Toda descripción en CLASSIC se puede expresar mediante lógica de primer orden, pero algunas descripciones resultan más directas expresadas en CLASSIC. Por ejemplo, para describir el conjunto de hombres que por lo menos tengan tres hijos, estén desempleados y cuya esposa es doctora, y que además tengan como máximo dos hijas, ambas profesoras de Física o Matemáticas se escribiría

$$\begin{aligned}&Y(Hombre, AlMenos(3, Hijos), AlMenos(2, Hijas),\\&\quad Todos(Hijos, Y(Desempleado, Casado, Todos(Esposa, Doctor))),\\&\quad Todos(Hijas, Y(Profesor, Satisface(Departamento, Física, Matemáticas))))\end{aligned}$$

Se deja como ejercicio traducir lo anterior a lógica de primer orden.

Quizás el aspecto más importante de la lógica descriptiva sea el énfasis que se pone en la maleabilidad de la inferencia. Los problemas se resuelven mediante su descripción y cuestionando si pueden ser subsumidos mediante una de las varias categorías posibles de solución. En los sistemas promedio de lógica de primer orden, muchas veces es imposible predecir cuál va a ser el tiempo necesario para hallar la solución. Frecuentemente, se deja al usuario diseñar la representación que permita evitar aquellos conjuntos de oraciones, que probablemente sean las causantes de que el sistema emplee varias semanas en resolver un problema. El énfasis en la lógica descriptiva, por otra parte, es el de ga-

[9] Obsérvese que el lenguaje *no* permite limitarse a afirmar que un concepto o una categorías es subconjunto de otro. Lo anterior es deliberativo: la subsunción entre categorías debe obtenerse a partir de ciertos aspectos de las descripciones de las categorías. De no ser así, seguramente algo está faltando en las descripciones.

Concepto	→	**Objeto** | *NombreDelConcepto*
	|	**Y**(*Concepto*, ...)
	|	**Todo**(*NombreRol*, *Concepto*)
	|	**AlMenos**(*Entero*, *NombreRol*)
	|	**Casi**(*Entero*, *NombreRol*)
	|	**Cumple**(*NombreRol*, *NombreIndividual*, ...)
	|	**IgualQue**(*Ruta*, *Ruta*)
	|	**UnoDe**(*NombreIndividual*, ...)
Ruta	→	[*NombreDelRol*, ...]

Figura 10.11 La sintaxis de descripciones en un subconjunto del lenguaje CLASSIC.

rantizar que la prueba de subsunción pueda ser resuelta en un tiempo que sea una función polinómica del tamaño de la descripción del problema[10].

En principio, lo anterior parecería maravilloso hasta que uno se da cuenta que entraña dos consecuencias: los problemas difíciles no se pueden formular, o ¡requieren de descripciones de vastas extensiones exponenciales! A pesar de lo anterior, la docilidad de los resultados obtenidos arrojan luz sobre qué tipos de estructuras causan problemas, y esto ayuda al usuario a comprender el comportamiento de las diversas representaciones.

Por ejemplo, la lógica descriptiva por lo general carece de *negación* y *disyunción*. Ambas fuerzan a los sistemas lógicos de primer orden a producir análisis de caso de tipo exponencial, si es que se desea garantizar la completitud. Por la misma razón se les ha excluido de Prolog. CLASSIC acepta sólo una variante limitada de la disyunción, las estructuras *SatisfaceQue* y *UnaDe*, que permiten aplicar la disyunción en individuos específicamente designados, pero no en las descripciones. En el caso de las descripciones disyuntivas, las definiciones anidadas dan lugar fácilmente a una cantidad exponencial de rutas alternativas, en las que una categoría puede subsumir a otra.

10.7 Razonamiento con información por defecto

En la sección precedente, se ha visto un ejemplo simple de una aserción con un valor por defecto: las personas tienen dos piernas. Este valor por defecto puede ser sobrescrito con información más específica, como que Long John Silver tiene una pierna. Se ha visto que el mecanismo de herencia en las redes semánticas implementa la sobrescritura de valores por defecto de forma simple y natural. En esta sección, se estudiarán los valores por defecto de forma más general, con la vista puesta hacia la comprensión de la *semántica* de los valores por defecto, en vez de proporcionar tan sólo un mecanismo procedimental.

[10] CLASSIC proporciona una prueba de subsunción eficiente en la práctica, pero el peor caso de ejecución es exponencial.

Mundos abiertos y cerrados

Suponga que estaba mirando en un tablón de noticias en el Departamento de Informática de una Universidad y vio una nota que decía, «Se ofrecerán los siguientes cursos: CS 101, CS 102, CS 106, EE 101». ¿Cuántos cursos se ofrecerán?, si su respuesta es «cuatro», estará en consonancia con un sistema de base de datos típico. Dada una base de datos relacional con el equivalente de las cuatro aserciones

$$Curso(CS, 101), Curso(CS, 102), Curso(CS, 106), Curso(EE, 101), \qquad (10.2)$$

La consulta SQL `count * from Curso` devuelve 4. Por otro lado, un sistema basado en lógica de primer orden respondería «algo entre uno e infinito», no «cuatro». La razón es que las aserciones *Curso* no niegan la posibilidad de que se impartan otros cursos no mencionados, ni que los cursos que se mencionan sean diferentes los unos de los otros.

Este ejemplo muestra que los sistemas de bases de datos y las convenciones de los humanos para la comunicación, difieren de la lógica de primer orden en al menos dos puntos. En primer lugar, las bases de datos (y la gente) asume que la información proporcionada es *completa*, por lo que las sentencias atómicas para las que no se dispone de una aserción que diga que son ciertas, se consideran falsas. Esto es lo que se conoce con el nombre de **asunción del mundo cerrado**, o CWA (*Closed-World Assumption*). En segundo lugar, normalmente se asume que nombres distintos hacen referencia a objetos distintos. Esto se conoce con el nombre de **asunción de nombres únicos**, o UNA (*Unique Names Assumption*), introducido primero en el contexto de nombres de acciones en la Sección 10.3.

ASUNCIÓN DEL MUNDO CERRADO

La lógica de primer orden no asume estas convenciones, y por lo tanto, necesita ser más explícita.

Para decir que *sólo* se ofertan los cuatros cursos distintos, se podría escribir:

$$\begin{aligned} Curso(d, n) \quad \Leftrightarrow \quad & [d, n] = [CS, 101] \vee [d, n] = [CS, 102] \\ & \vee [d, n] = [CS, 106] \vee [d, n] = [EE, 101] \end{aligned} \qquad (10.3)$$

COMPLETITUD

La Ecuación (10.3) se llama de **completitud**[11] de 10.2. En general, la completitud contendrá una definición (una sentencia si y sólo si) para cada predicado, y cada definición contendrá una disyunción por cada cláusula definida, teniendo el predicado a la cabeza[12]. En general, la completitud se construye como sigue:

1. Unir todas las cláusulas con el mismo nombre de predicado (P) y la misma cardinalidad (n).
2. Transformar cada cláusula en **forma normal de Clark**: reemplazar

FORMA NORMAL DE CLARK

$$P(t_1 ..., t_n) \leftarrow Cuerpo$$

donde t_i son términos, con

$$P(v_1 ..., v_n) \leftarrow \exists\, w_1 \ldots w_m\ [v_1 \ldots, v_n] = [t_1 ..., t_n] \wedge Cuerpo$$

[11] Algunas veces llamada «Completidud de Clark» debido a su creador, Keith Clark.
[12] Nótese que esta es también la forma de los axiomas estado-sucesor vistos en la Sección 10.3.

donde v_i son variables inventadas recientemente y w_i son las variables que aparecen en la cláusula original. Usar el mismo conjunto de v_i para cada cláusula. Esto proporciona el siguiente conjunto de cláusulas

$$P(v_1 ..., v_n) \leftarrow B_1$$
$$\vdots$$
$$P(v_1 ..., v_n) \leftarrow B_k$$

3. Combinar todo esto junto a una cláusula disyuntiva:

$$P(v_1 ..., v_n) \leftarrow B_1 \vee ... \vee B_k$$

4. Formar la completitud reemplazando el $\leftarrow$ con la equivalencia:

$$P(v_1 ..., v_n) \Leftrightarrow B_1 \vee ... \vee B_k$$

La Figura 10.12 muestra un ejemplo de la completitud de Clark para una base de conocimiento con hechos y reglas. Para añadir la asunción de nombres únicos, simplemente se construye la completitud de Clark para la relación de igualdad, donde sólo se conocen los hechos $CS = CS$, $101 = 101$, etc. Esto se deja como un ejercicio.

La asunción del mundo cerrado nos permite encontrar un **modelo mínimo** de una relación. Es decir, se puede encontrar el modelo de la relación *Curso* con los menores elementos. En la Ecuación (10.2) el modelo mínimo de *Curso* tiene cuatro elementos. Alguno menos y existiría una contradicción. Para las bases de conocimiento de Horn, siempre hay un modelo mínimo *único*. Nótese que la asunción de nombres únicos también se aplica a la relación de igualdad: cada término es igual sólo a él mismo. Paradójicamente, esto significa que los modelos mínimos son máximos en el sentido de tener tantos objetos como sea posible.

Cláusulas de Horn	**Completitud de Clark**
$Curso(CS, 101)$	$Curso(d, n) \Leftrightarrow [d, n] = [CS, 101]$
$Curso(CS, 102)$	$\vee [d, n] = [CS, 102]$
$Curso(CS, 106)$	$\vee [d, n] = [CS, 106]$
$Curso(EE, 101)$	$\vee [d, n] = [EE, 101]$
$Curso(EE, i) \leftarrow Entero(i)$	$\vee \exists\, i\, [d, n] = [EE, i] \wedge Entero(i)$
$\wedge\, 101 \leq i \wedge i \leq 130$	$\wedge\, 101 \leq i \wedge i \leq 130$
$Curso(CS, m + 100) \Leftarrow$	$\vee \exists\, m\, [d, n] = [CS, m + 100]$
$Curso(CS, m) \wedge 100 \leq m$	$\wedge\, Curso(CS, m) \wedge 100 \leq m$
$\wedge\, m < 20$	$\wedge\, m < 200$

Figura 10.12 La Completitud de Clark usa un conjunto de cláusulas de Horn. El programa original de Horn (izquierda), lista explícitamente cuatro cursos y también afirma que hay clase de matemáticas para cada entero desde 101 hasta 130, y que por cada clase CS en la serie 100 (no titulados) hay una clase correspondiente en la serie 200 (titulados). La completitud de Clark (derecha) establece que no hay otras clases. Con la completitud y la asunción de nombres únicos (así como la definición obvia del predicado *Entero*), se consigue la conclusión deseada de la existencia de exactamente 36 cursos: 30 de matemáticas y seis de tipo CS.

Es posible coger un programa de Horn, generar la completitud de Clark y pasar el resultado obtenido a un demostrador de teoremas para realizar inferencia. Pero a menudo es más eficiente utilizar un mecanismo de inferencia de propósito específico como es Prolog, que tiene implementado en el mecanismo de inferencia las asunciones del mundo cerrado y nombres únicos.

Aquellos sistemas que implementan la asunción del mundo cerrado deben ser cuidadosos sobre el tipo de razonamiento que llevarán a cabo. Por ejemplo, en una base de datos del censo, sería razonable asumir la asunción del mundo cerrado cuando se razone sobre la población actual de las ciudades, pero sería erróneo deducir que ningún niño nacería en el futuro porque la base de datos no contenga entradas con futuros nacimientos. La asunción del mundo cerrado hace que la base de datos sea **completa**, en el sentido de que cualquier consulta atómica se responde positiva o negativamente. Cuando se es genuinamente ignorante acerca de los hechos (como futuros nacimientos), no se puede usar la asunción del mundo cerrado. Un sistema de representación del conocimiento más sofisticado podría permitir al usuario especificar reglas para decidir cuándo aplicar la asunción del mundo cerrado.

Negación como fallo y semánticas de modelado estables

Se ha visto en los Capítulos 7 y 9 que las bases de conocimiento en forma de Horn tienen propiedades computacionalmente deseables. En muchas aplicaciones, sin embargo, el requisito de que cada literal en el cuerpo de una cláusula debe ser positivo es un inconveniente grande. Se podría querer afirmar «Tú puedes salir si no está lloviendo», sin tener que crear predicados como *NoLloviendo*. En esta sección, se verá una forma de añadir una negación explícita a las cláusulas de Horn usando la idea de **negación como fallo**. La idea es que un literal negativo, *no P*, puede ser «demostrado» cierto, en el caso de que la prueba de *P* falle. Esta es una forma de razonamiento por defecto relacionada íntimamente con la asunción del mundo cerrado: se asumirá que algo es falso si no puede ser demostrado que sea cierto. Se usará «*no*» para distinguir la negación como fallo del operador lógico «¬».

NEGACIÓN COMO FALLO

Prolog permite el operador *no* en el cuerpo de una cláusula. Por ejemplo, considérese el siguiente programa escrito en Prolog:

$$\begin{aligned} &dispositivoIDE \leftarrow Dispositivo \wedge no\ dispositivoSCSI \\ &dispositivoSCSI \leftarrow Dispositivo \wedge no\ dispositivoIDE \\ &controladorSCSI \leftarrow dispositivoSCSI \\ &Dispositivo \end{aligned} \tag{10.4}$$

La primera regla dice si se dispone de un disco duro en un computador y no es SCSI, entonces debe ser IDE. La segunda establece que si no es IDE debe ser SCSI. La tercera establece que tener una unidad SCSI implica tener un controlador SCSI y la cuarta afirma que efectivamente se dispone de una unidad. Este programa tiene dos modelos mínimos:

$$\begin{aligned} M_1 &= \{Dispositivo, dispositivoIDE\} \\ M_2 &= \{Dispositivo, dispositivoSCSI, controladorSCSI\} \end{aligned}$$

Los modelos mínimos no pueden capturar la semántica específica de programas que emplean negación como fallo. Considérese el programa

$$P \leftarrow no\ Q \tag{10.5}$$

Tiene dos modelos mínimos, $\{P\}$ y $\{Q\}$. Desde el punto de vista de la lógica de primer orden esto tiene sentido, puesto que $P \Leftarrow \neg Q$ es equivalente a $P \vee Q$. Pero desde el punto de vista de Prolog es inquietante: Q nunca aparece en la parte izquierda de una flecha, por lo que ¿cómo puede ser un consecuente?

MODELO ESTABLE

JUSTIFICACIÓN

REDUCTO

Una alternativa es la idea de **modelo estable**, que es un modelo mínimo donde cada átomo en el modelo tiene una **justificación**: una regla en la cual la cabeza es el átomo y cada literal en el cuerpo se satisface. Técnicamente, se dice que M es un modelo estable de un programa H, si M es el único modelo único del **reducto** de H con respecto a M. El reducto de un programa H se obtiene borrando de H cualquier regla que tenga un literal *no A* en el cuerpo, donde A está en el modelo, y posteriormente borrando cualquier literal negativo en las restantes reglas. Puesto que el reducto de H es ahora una lista de cláusulas de Horn, debe tener un único modelo mínimo.

El reducto de $P \leftarrow no\ Q$ con respecto a $\{P\}$ es P, que tiene como modelo mínimo $\{P\}$. Por lo tanto $\{P\}$ es un modelo estable. El reducto con respecto a $\{Q\}$ es el programa vacío, que tiene como modelo mínimo $\{\ \}$. Por lo tanto $\{Q\}$ no es un modelo estable porque Q no tiene justificación en la Ecuación (10.5). Como otro ejemplo, el reducto de (10.4) con respecto a M_1 es como sigue:

$$\begin{array}{l} dispositivoIDE \leftarrow Dispositivo \\ controladorSCSI \leftarrow dispositivoSCSI \\ Dispositivo \end{array}$$

PROGRAMACIÓN DE CONJUNTO DE RESPUESTAS

CONJUNTO DE RESPUESTAS

Tiene como modelo mínimo M_1, por lo tanto M_1 es un modelo estable. La **programación de conjunto de respuestas** es una clase de lógica de programación que incorpora negación como fallo y funciona transformando la lógica del programa en forma base, para buscar modelos estables (también conocidos como **conjunto de respuestas**) a través del uso de técnicas de prueba de modelos proposicionales. Por lo tanto, la programación de conjunto de respuestas es un descendente del Prolog y los demostradores rápidos de satisfacción proposicional como WALKSAT. De hecho, la programación de conjunto de respuestas ha sido aplicada con éxito a problemas de planificación, de igual forma que los demostradores rápidos de satisfacción proposicional. La ventaja de la planificación de conjunto de respuestas sobre otros planificadores es el nivel de flexibilidad: los operadores de planificación y las restricciones se pueden expresar como programas lógicos, y no están restringidos a un formato específico de un formalismo planificador particular. La desventaja es la misma que para otras técnicas proposicionales: si hay muchos objetos en el universo, entonces puede tener lugar una ralentización exponencial.

Circunscripción y lógica por defecto

Se han visto dos ejemplos donde, de forma aparente, el proceso natural de razonamiento violaba la propiedad monotónica de la lógica, que fue demostrada en el Capítulo 7[13]. En el primer ejemplo, una propiedad heredada por todos los miembros de una catego-

[13] Recuérdese que la monotonía requiere que en todas las sentencias de implicación se mantenga la implicación después de que se añadan nuevas sentencias a la base de conocimiento (KB). Es decir, si $KB \models \alpha$ entonces $KB \wedge \beta \models \alpha$.

ría en una red semántica, podía ser sobrescrita por información específica de una subcategoría. En el segundo ejemplo, los literales negados que derivan de la asunción del mundo cerrado podrían se sobrescritos por la adición de literales positivos.

La introspección simple sugiere que esos fallos de la propiedad monotónica son generalizados en el razonamiento basado en el sentido común. Parece que los humanos a menudo «llegan a conclusiones». Por ejemplo, cuando uno ve un coche aparcado en la calle, uno tiende normalmente a creer que el coche tiene cuatro ruedas aunque sólo sean visibles tres (si usted cree que la existencia de la cuarta rueda es dudosa, considere también la cuestión de si las tres ruedas visibles son reales o simplemente unas maquetas). Ahora, la teoría de la probabilidad puede proporcionar una conclusión de que la cuarta rueda existe con una alta probabilidad, incluso, para la mayoría de la gente, la posibilidad de que el coche no tenga la cuarta rueda *no surge a menos que se presente alguna nueva evidencia*. Por lo tanto, parece que la conclusión de la cuarta rueda se alcanza *por defecto*, en ausencia de otra razón que lo ponga en duda. Si se detectan nuevas evidencias (por ejemplo, si uno ve al propietario del coche llevando una rueda y se da cuenta de que el coche tiene un gato), entonces la conclusión puede ser negada. Esta clase de razonamiento se dice que es **no monotónico**, porque el conjunto de creencias no crece monotónicamente con el tiempo cuando se dispone de nuevo conocimiento. La **lógica no monotónica** ha sido concebida con nociones modificadas de verdad e implicación para capturar este comportamiento. Se examinarán estos dos tipos de lógica que han sido estudiados de forma extensiva: circunscripción y lógica por defecto.

NO MONOTÓNICO

LÓGICA NO MONOTÓNICA

CIRCUNSCRIPCIÓN

La **circunscripción** puede ser vista como una versión más potente y precisa de la asunción del mundo cerrado. La idea es especificar predicados particulares que son asumidos como «tan falsos como posibles» (es decir, falso para todo objeto excepto para aquellos para los cuales se conoce que es cierto). Por ejemplo, supóngase que se quiere expresar la regla por defecto de que los pájaros vuelan. Se podría introducir un predicado, por ejemplo $Anormal_1(x)$, y escribir

$$Pájaro(x) \wedge \neg Anormal_1(x) \Rightarrow Vuela(x)$$

Si se especifica que $Anormal_1$ va a ser **circunscrito**, un razonador que utiliza circunscripción asumirá $\neg Anormal_1(x)$ a menos que sea conocida la certeza de $Anormal_1(x)$. Esto proporciona que la conclusión $Vuela(Tweety)$ se pueda alcanzar a partir de la premisa $Pájaro(Tweety)$, pero la conclusión no se mantendrá si se afirma que $Anormal_1(Tweety)$.

PREFERENCIA DE MODELOS

La circunscripción puede ser vista como un ejemplo de lógica de **preferencia de modelos**. En este tipo de lógica, una sentencia se deduce (con valor por defecto) si es cierta en todos los modelos *preferidos* de la base de conocimiento, en contraposición a los requerimientos de verdad en *todos* los modelos de la lógica clásica. Por circunscripción, un modelo se prefiere a otro si tiene menos objetos anormales[14]. Veamos cómo trabaja esta idea en el contexto de la herencia múltiple en las redes semánticas. El ejemplo estándar para el cual la herencia múltiple genera problemas se llama «El diamante de Nixon».

[14] En la asunción del mundo cerrado, se prefiere un modelo a otro si tiene menos átomos ciertos (es decir, se prefieren modelos que son **mínimos**). Existe una conexión natural entre la asunción del mundo cerrado y las cláusulas definidas de las bases de conocimiento, porque el punto fijo alcanzado mediante razonamiento hacia delante en estas bases de conocimiento es el modelo mínimo único (*véase* Apartado 7.5).

Surge de la observación de que Richard Nixon es un cuáquero (y por lo tanto pacifista) y un republicano (y por lo tanto no pacifista). Se podría escribir esto como sigue:

$$Republicano(Nixon) \wedge Cuáquero(Nixon)$$
$$Republicano(x) \wedge \neg Anormal_2(x) \Rightarrow \neg Pacifista(x)$$
$$Cuáquero(x) \wedge \neg Anormal_3(x) \Rightarrow Pacifista(x)$$

Si se circunscriben $Anormal_2$ y $Anormal_3$, no existen modelos preferidos: uno en el que se cumpla $Anormal_2(Nixon)$ y $Pacifista(Nixon)$ se considera válido, y otro en el que se cumpla $Anormal_3(Nixon)$ y $\neg Pacifista(Nixon)$ también se considera válido. Por lo tanto, el razonador basado en circunscripción permanece agnóstico a la idea de que Nixon es pacifista. Si se desea además afirmar que las creencias religiosas tienen precedencia sobre creencias políticas, se podría usar un formalismo denominado **circunscripción priorizada** para dar preferencia a modelos donde $Anormal_3$ es minimizado.

CIRCUNSCRIPCIÓN PRIORIZADA

LÓGICA POR DEFECTO

REGLAS POR DEFECTO

La **lógica por defecto** es un formalismo en el cual las **reglas por defecto** se pueden escribir para generar conclusiones contingentes no monotónicas. Una regla por defecto se parece a la siguiente:

$$Pájaro(x) : Vuela(x) \,/\, Vuela(x)$$

Esta regla significa que si $Pájaro(x)$ es cierto y si $Vuela(x)$ es consistente con la base de conocimiento, entonces se puede concluir $Vuela(x)$ por defecto. En general, una regla por defecto tiene la forma

$$P : J_1 ..., J_n \,/\, C$$

donde P es un prerrequisito, C es la conclusión, y J_i son las justificaciones (si cualquiera de ellas se puede probar como falsa, entonces la conclusión no se puede alcanzar). Cualquier variable que aparece en J_i o C también debe aparecer en P. El ejemplo del diamante de Nixon puede ser representado utilizando lógica por defecto con un hecho y dos reglas por defecto:

$$Republicano(Nixon) \wedge Cuáquero(Nixon)$$
$$Republicano(x) : \neg Pacifista(x) \,/\, \neg Pacifista(x)$$
$$Cuáquero(x) : Pacifista(x) \,/\, Pacifista(x)$$

EXTENSIÓN

Para interpretar cuál es el significado de las reglas por defecto, se definirá la noción de una **extensión** o una teoría por defecto como el conjunto máximo de consecuentes de la teoría. Es decir, una extensión S consta de los hechos originales conocidos y un conjunto de conclusiones a partir de las reglas por defecto, de tal modo que no se pueden alcanzar conclusiones adicionales desde S y las justificaciones de cada conclusión por defecto en S son consistentes con S. Como en el caso de los modelos preferidos en la circunscripción, se tienen dos posibles extensiones para el diamante de Nixon: uno en el que él es un pacifista y otro en el que no lo es. Existen esquemas priorizados en los que a algunas reglas por defecto se les puede dar precedencia sobre otras, permitiendo resolver algunas ambigüedades.

Desde 1980, cuando se propuso por primera vez la lógica no monotónica, se han hecho muchos progresos para entender sus propiedades matemáticas. Comenzando en los últimos años de la década de los 90, los sistemas prácticos basados en programación ló-

gica han demostrado ser prometedores como herramientas para la representación del conocimiento. Sin embargo, aún hay cuestiones por resolver. Por ejemplo, si «los coches tienen cuatro ruedas» es falso, ¿qué significa el que lo tengamos en nuestra base de conocimiento? ¿Cuál es el conjunto apropiado de reglas por defecto que debemos tener? Si no se puede decidir, para cada regla por separado, si debe pertenecer a la base de conocimiento, entonces existe un problema serio de no modularidad. Por último, ¿cómo se pueden usar las creencias que tienen un valor por defecto para tomar decisiones? Probablemente esta sea la cuestión más difícil de resolver para el razonamiento por defecto. Las decisiones a menudo implican cambios, y por lo tanto, se necesita comparar la *fuerza* en la creencia de la salida de varias acciones. En aquellos casos donde se toma repetidamente la misma clase de decisiones, es posible interpretar las reglas por defecto como afirmaciones que establecen un «umbral de probabilidad». Por ejemplo, la regla por defecto «mis frenos están siempre correctos» realmente significa «La probabilidad de que mis frenos estén correctos, en ausencia de otra información, es suficientemente alta para que la decisión óptima sea conducir sin revisarlos». Cuando el contexto para la toma de decisiones cambia (por ejemplo, cuando uno conduce un camión pesado muy cargado por una carretera de montaña empinada), el valor por defecto de repente no es el apropiado, incluso cuando no haya evidencia que sugiera que los frenos estén mal. Estas consideraciones han llevado a algunos investigadores a considerar cómo encapsular el razonamiento por defecto en la teoría de la probabilidad.

10.8 Sistemas de mantenimiento de verdad

La sección anterior argumentaba que la mayoría de las inferencias logradas por un sistema de representación del conocimiento tendrán sólo valores por defecto, en vez de ser absolutamente ciertas. Inevitablemente, algunos de los hechos inferidos serán erróneos y tendrán que ser retractados debido a la aparición de nueva información. Este proceso de llama **revisión de la creencia**[15]. Supóngase que una base de conocimiento *KB* contiene una sentencia *P* (puede ser una conclusión por defecto generada por un algoritmo de razonamiento hacia delante, o simplemente una afirmación incorrecta) y que se desea ejecutar *Decir*(*KB*, ¬*P*). Para evitar el crear una contradicción, primero se ejecutará *Retractar*(*KB*, *P*). Esto parece sencillo, sin embargo, el problema surge si alguna sentencia *adicional* fue inferida utilizando *P* y afirmada en la base de conocimiento. Por ejemplo, la implicación $P \Rightarrow Q$ podría haber sido utilizada para añadir *Q*. La solución «obvia» (retractar todas las sentencias que se infirieron utilizando *P*) falla, porque esas sentencias podrían tener otras justificaciones a demás de *P*. Por ejemplo, si también están en la base de conocimiento *R* y $R \Rightarrow Q$, entonces *Q* no debe ser borrado. Los **sistemas de mantenimiento de verdad**, o SMV, están diseñados para manejar justamente este tipo de complicaciones.

REVISIÓN DE LA CREENCIA

SISTEMAS DE MANTENIMIENTO DE VERDAD

[15] La revisión de la creencia se compara a menudo con la **actualización de la creencia**, que ocurre cuando se revisa una base de conocimiento para reflejar un cambio en el mundo, en lugar de trabajar con nueva información de un mundo fijo. La actualización de la creencia combina revisión de la creencia con razonamiento sobre el tiempo y el cambio. También está relacionada con el proceso de **filtrado** descrito en el Capítulo 15.

Una aproximación muy simple al mantenimiento de verdad es mantener la pista del orden en el que las sentencias se introducen en la base de conocimiento numerándolas desde P_1 hasta P_n. Cuando se realiza una llamada a *Retractar*(KB, P_i), el sistema involuciona hasta el estado anterior en el que P_i fue añadido, eliminando de ese modo P_i y cualquier inferencia que fuera derivada de P_i. Las sentencias P_{i+1} hasta P_n pueden ser añadidas de nuevo. Esto es simple y garantiza que la base de conocimiento será consistente, pero el hecho de retractar P_i requiere retractar y afirmar $n - i$ sentencias, así como deshacer y rehacer todas la inferencias alcanzadas utilizando esas sentencias. En los sistemas donde se añaden mucho hechos (como bases de batos comerciales) esto es impracticable.

SMVJ

JUSTIFICACIÓN

Una aproximación más eficiente son los sistemas de mantenimiento de verdad basados en justificación, o **SMVJ**. En un SMVJ, cada sentencia en la base de conocimiento se anota con una **justificación**, que consiste en el conjunto de sentencias a partir de las cuales fue inferida. Por ejemplo, si la base de conocimiento actualmente contiene $P \Rightarrow Q$, entonces DECIR(P) causará que se añada Q con la justificación $\{P, P \Rightarrow Q\}$. En general, una sentencia puede tener cualquier número de justificaciones. Las justificaciones se utilizan para que la operación de retractar sea eficiente. Dada la invocación RETRACTAR(P), el SMVJ detectará exactamente aquellas sentencias que tienen P como una justificación. Por lo tanto, si una sentencia Q tiene como única justificación $\{P, P \Rightarrow Q\}$, será borrada. Si además también tuviera la justificación $\{P, P \vee R \Rightarrow Q\}$, también sería borrada, pero si tuviera la justificación $\{R, P \vee R \Rightarrow Q\}$, entonces no sería borrada. De esta forma, el tiempo necesario para retractar P depende sólo del número de sentencias derivadas de P, en lugar de depender del número de sentencias añadidas desde que se insertó P en la base de conocimiento.

Los SMVJ asumen que las sentencias que han sido consideradas una vez, serán consideradas de nuevo, por lo tanto en lugar de borrar una sentencia de la base de conocimiento cuando pierde todas sus justificaciones, simplemente se marcarán las sentencias como si estuvieran *fuera* de la base de conocimiento. Si una afirmación posterior restituye una de las justificaciones, entonces la sentencia se marca de nuevo como *dentro*. De esta forma, los SMVJ mantienen toda la cadena de inferencia que utilizan y no vuelven a inferir sentencias cuando una justificación vuelve a ser válida.

Además de manejar el proceso de retractar información incorrecta, los SMV se pueden utilizar para acelerar el análisis de múltiples situaciones hipotéticas. Supóngase, por ejemplo, que el Comité Olímpico Romano está seleccionando lugares para natación, competiciones atléticas y eventos ecuestres para los Juegos Olímpicos que se celebrarán en Roma en el año 2048. Por ejemplo, sea la primera hipótesis *Lugar*(*Natación*, *Pitesti*), *Lugar*(*CompeticionesAtléticas*, *Bucarest*) y *Lugar*(*EventosEcuestres*, *Arad*). Se debe realizar una gran cantidad de razonamiento para obtener las consecuencias logísticas y por lo tanto la bondad de esta selección. Si en su lugar se desea considerar *Lugar*(*CompeticionesAtléticas*, *Sibiu*), el SMV no tiene que empezar otra vez desde cero. En su lugar, simplemente se retractará *Lugar*(*CompeticionesAtléticas*, *Bucarest*) y se afirmará *Lugar*(*CompeticionesAtléticas*, *Sibiu*) y el SMV llevará a cabo las revisiones necesarias. La cadena de inferencia generada por la opción de Bucarest se puede reutilizar con Sibiu, dado que la conclusión es la misma.

SMVS

Un sistema de mantenimiento de verdad basado en suposiciones, o **SMVS** está diseñado para realizar este tipo de cambios de contexto entre mundos hipotéticos de un

modo muy eficiente. En un SMVJ, el mantenimiento de las justificaciones permite moverse rápidamente de un estado a otro, realizando pocas retracciones y afirmaciones, pero en un momento dado sólo se puede representar un estado. Un SMVS representa *todos* los estados que han sido considerados al mismo tiempo. Mientras que un SMVJ simplemente etiqueta cada sentencia como *dentro* o *fuera*, el SMVS lleva el control, por cada sentencia, de qué suposiciones harán que la sentencia sea verdadera. Es decir, cada sentencia tiene una etiqueta que está formada por un conjunto de suposiciones. La sentencia se cumple sólo en aquellos casos en los que todas las suposiciones de uno de los conjuntos de suposición se cumple.

EXPLICACIONES

SUPOSICIONES

Los sistemas de mantenimiento de verdad también proporcionan un mecanismo para generar **explicaciones**. Técnicamente, una explicación de una sentencia P es un conjunto de sentencias E de modo que E implica P. Si se conoce que las sentencias en E son ciertas, entonces E simplemente proporciona una base suficiente para probar que P debe ser el caso. Pero las explicaciones también incluyen **suposiciones** (sentencias que no se sabe si son ciertas, pero en el caso de serlo serían suficientes para probar P). Por ejemplo, uno puede no tener suficiente información para probar que su coche no arrancará, pero una explicación razonable podría incluir la suposición de que la batería está agotada. Esto, combinado con el conocimiento de cómo funciona un coche, explica el hecho de que no arranque. En la mayoría de los casos, se prefiere una explicación E que es mínima, con el significado de que no existe un subconjunto propio de E que sea también una explicación. Un SMVS puede generar explicaciones para el problema de que «el coche no arranca» a través de la realización de suposiciones (como «combustible en el coche» o «batería agotada») en el orden que se desee, incluso si algunas suposiciones son contradictorias. A continuación se comprueba la etiqueta de la sentencia «el coche no arranca», para verificar el conjunto de suposiciones que justificarían la sentencia.

Los algoritmos empleados en la implantación de los sistemas de mantenimiento de verdad son un poco complicados y no se comentarán aquí. La complejidad computacional del problema de mantenimiento de verdad es por lo menos tan grande como la de la inferencia proposicional, es decir, de dificultad NP. Por ello, no es de esperar que el mantenimiento de verdad sea una panacea. Sin embargo, cuando se utiliza con cuidado un SMV puede proporcionar un incremento sustancial en la habilidad de un sistema lógico para manejar entornos complejos e hipótesis.

10.9 Resumen

Este capítulo ha sido con diferencia el más detallado del libro. Ahondando en los detalles de cómo uno representa una gran variedad de conocimiento, se espera haber dado al lector una idea de cómo se construyen las bases de conocimiento reales. Los puntos más importantes son los siguientes:

- La representación de conocimiento a gran escala necesita una ontología de propósito general para organizar y unir varios dominios de conocimiento específicos.
- Una ontología de propósito general necesita abarcar una amplia gama de conocimiento y debería ser capaz, en principio, de manejar cualquier dominio.

- Se ha presentado una **ontología superior** basada en categorías y en el cálculo de eventos. Se ha cubierto la estructura de objetos, espacio y tiempo, cambio, procesos, sustancias y creencias.
- Las acciones, los eventos y el tiempo se pueden representar utilizando el cálculo de situaciones o una representación más expresiva como el cálculo de eventos y el cálculo de flujos. Estas representaciones capacitan a un agente para construir planes mediante inferencia lógica.
- Los estados mentales de los agentes se pueden representar mediante cadenas que denoten creencias.
- Se ha presentado un análisis detallado del dominio de compra a través de Internet, ejercitando la ontología general y mostrando cómo el conocimiento del domino se puede utilizar en un agente de compra.
- Los sistemas de representación de propósito específico, como las **redes semánticas** y la **lógica descriptiva**, han sido concebidos para ayudar en la organización jerárquica de categorías. La **herencia** es una forma importante de inferencia, permitiendo deducir las propiedades de los objetos a partir de su pertenencia a categorías.
- La **asunción del mundo cerrado**, implementada en programa lógicos, proporciona una forma simple de evitar el tener que especificar montones de información negativa. Es mejor su representación como **información por defecto** que puede ser sobrescrita por información adicional.
- La **lógica no monotónica**, como la **circunscripción** y la **lógica por defecto**, se utiliza para capturar el razonamiento por defecto en general. La **programación de conjunto de respuestas** acelera la inferencia no monotónica, así como WALKSAT acelera la inferencia proposicional.
- Los **sistemas de mantenimiento de verdad** manejan las actualizaciones y revisiones del conocimiento de forma eficiente.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

Existen afirmaciones plausibles (Briggs, 1985) en el sentido de que la investigación formal sobre la representación del conocimiento se inició en la India con los trabajos teóricos sobre la gramática del sánscrito shástrico, que datan del primer milenio a.C. En Occidente, el primer ejemplo lo constituyó el empleo de las definiciones de términos de la antigua matemática griega. De hecho, el desarrollo de terminología técnica en cualquier campo puede ser entendido como una forma de representación del conocimiento.

Los debates iniciales acerca de la representación en IA se enfocaban hacia la «representación del *problema*» en lugar de la «representación del *conocimiento*» (*véase*, por ejemplo, La discusión de Amarel (1968) acerca del problema de los caníbales y los misioneros). En los años 70, la IA hizo énfasis en el desarrollo de «sistemas expertos» (también llamados «sistemas basados en conocimiento») que podían, dado un conocimiento apropiado del dominio, alcanzar o superar el rendimiento de expertos humanos en tareas

específicas. Por ejemplo, el primer sistema experto, DENDRAL (Feigenbaum *et al*., 1971; Lindsay *et al*., 1980), interpretó la salida de un espectómetro de masas (un instrumento usado para analizar la estructura de compuestos químicos orgánicos) de forma más precisa que un experto químico. Si bien el éxito obtenido por DENDRAL jugó un papel decisivo para que la comunidad de investigadores en IA tomara conciencia de la importancia que revestía la representación del conocimiento, los formalismos de representación utilizados en DENDRAL son muy específicos para el dominio de la química. Durante tiempo, los investigadores se interesaron en formalismos para la representación estandarizada del conocimiento y en ontologías que pudieran hacer más eficiente el proceso de crear nuevos sistemas expertos. Fue así como se aventuraron en un territorio que anteriormente había sido explorado por los filósofos de la ciencia y del lenguaje. La disciplina impuesta en IA por la necesidad de probar que las propias teorías «funcionan», ha dado como resultado un progreso más rápido y sólido que cuando estos problemas eran dominio exclusivo de la Filosofía (si bien en ocasiones ha dado lugar a la reinvención de la rueda).

La creación de taxonomías entendibles o clasificaciones data de tiempos muy antiguos. Aristóteles (384-322 a.C.) hizo un fuerte énfasis en la clasificación y en esquemas de categorización. Su *Organon*, una colección de trabajos sobre lógica recapitulados por sus estudiantes después de su muerte, incluye un tratado titulado *Categorías*, en el cual intentó construir lo que nosotros hoy denominamos ontología superior. Él también introdujo la noción de **géneros** y **especies** para una clasificación de bajo nivel, aunque no con el significado moderno específico de biología. El sistema actual de clasificación biológica, incluyendo el uso de «nomenclatura binomial» (clasificación en géneros y especies en un sentido técnico), fue inventada por el biólogo sueco Carolus Linnaeus, o Carl von Linne (1707-1778). Los problemas asociados con las clases naturales y los límites de categorías inexactas han sido estudiados por Wittgenstein (1953), Quine (1953), Lakoff (1987) y Schwartz (1977) entre otros.

El interés en ontologías de gran escala es creciente. El proyecto CYC (Lenat, 1995; Lenat y Guha, 1990) ha creado una ontología superior de 6.000 conceptos con 60.000 hechos, y soporta una ontología global mucho mayor. El IEEE ha establecido el subcomité P1600.1, el grupo de trabajo de ontologías superiores estándar (*Standard Upper Ontology Working Group*), y la iniciativa de mente abierta (*Open Mind Initiative*) tiene afiliados más de 7.000 usuarios de Internet para introducir más de 40.000 hechos sobre conceptos relacionados con el sentido común. En la Web, estándares como RDF, XML y la Semántica de la Web (Berners-Lee *et al*., 2001) están emergiendo, aunque aún no son ampliamente utilizados. Las conferencias sobre *Ontologías Formales en Sistemas de Información* (FOIS, *Formal Ontology in Information Systems*), contienen artículos muy interesantes sobre las ontologías generales y específicas de un dominio.

La taxonomía utilizada en este capítulo ha sido desarrollada por los autores y está basada en parte en su experiencia en el proyecto CYC, así como en el trabajo de Hwang y Schubert (1993) y Davis (1990). Una debate inspirador acerca del proyecto general de la representación basada en el conocimiento del sentido común, aparece en el trabajo de Hayes *The Naive Physics Manifesto* (1978, 1985b).

La representación del tiempo, el cambio, las acciones y los eventos ha sido ampliamente estudiado en Filosofía y en Informática Teórica, así como en IA. La aproxi-

mación más antigua es la **lógica temporal**, que es una lógica especializada en la que cada modelo describe una trayectoria completa en el tiempo (usualmente lineal o ramificada), en vez de una estructura relacional estática. La lógica incluye operadores modales que se aplican a las fórmulas; $\Box p$ significa «p será siempre cierto» y $\Diamond p$ significa «p será cierto en algún momento en el futuro». El estudio de la lógica temporal fue iniciado por Aristóteles y las escuelas estoica y de Megara, en la antigua Grecia. En la actualidad, Findlay (1941) fue el primero en sugerir un cálculo formal para razonar sobre el tiempo, pero el trabajo de Arthur Prior (1967) es considerado el de mayor influencia. Los libros de texto incluyen las aproximaciones de Rescher y Urquhart (1971) y van Benthem (1983).

Los informáticos teóricos se han interesado desde hace tiempo en la formalización de las propiedades de los programas, vistas como secuencias de acciones computacionales. Burstall (1974) introdujo la idea de utilizar operadores modales para razonar acerca de los programas de computador. Poco después, Vaughan Pratt (1976) diseñó la **lógica dinámica**, en la que los operadores modales indican los efectos producidos por programas u otras acciones (*véase* también Harel, 1984). Por ejemplo, en la lógica dinámica, si α es el nombre de un programa, entonces «$[\alpha]p$» significaría «p será válida en todos los estados del mundo que son resultado de la ejecución del programa α en el actual estado del mundo» y «$\langle\alpha\rangle p$» significaría «p sería válida por lo menos en uno de los estados del mundo producido por la ejecución del programa α en el actual estado del mundo». Fischer y Ladner (1977) utilizaron la lógica dinámica en el análisis real de programas. Pnueli (1977) fue el primero en utilizar la lógica temporal clásica para razonar sobre los programas.

Mientras que la lógica temporal posiciona el tiempo directamente en la teoría modelo del lenguaje, las representaciones del tiempo en IA han tendido a incorporar axiomas sobre el tiempo y los eventos de forma explícita en la base de conocimiento, y no han dado al tiempo una posición especial en la lógica. Esta aproximación puede permitir una mayor claridad y flexibilidad en algunos casos. Además, el conocimiento temporal expresado utilizando lógica de primer orden se puede integrar más fácilmente con otro conocimiento que haya sido acumulado con esta notación.

El cálculo de situaciones de John McCarthy (1963), fue el primer tratamiento del tiempo y la acción en IA. El primer sistema de IA en hacer un uso sustancial del razonamiento de propósito general sobre acciones utilizando lógica de primer orden fue QA3 (Green, 1969b). Kowalski (1979b) desarrolló la idea de reificar proposiciones con el cálculo de situaciones.

El **problema del marco** fue reconocido por primera vez por McCarthy y Hayes (1969). Muchos investigadores consideraron el problema irresoluble utilizando lógica de primer orden, y esto estimuló un gran trabajo de investigación en lógica no monotónica. Los filósofos desde Dreyfus (1972) a Crockett (1994) han citado el problema del marco como un síntoma inevitable del fallo de toda la iniciativa de la IA. La solución parcial al problema de la representación del marco usando axiomas estado-sucesor se debe a Ray Reiter (1991). Una solución a la inferencia en el problema del marco fue hecha por el trabajo de Holldobler y Schneeberger (1990), que ha sido conocido como cálculo de flujos (Thielscher, 1999). El debate en este capítulo se basa parcialmente en el análisis realizado por Lin y Reiter (1997) y Thielscher (1999). Los libros de Shanahan

(1997) y Reiter (2001b) dan una visión completa y moderna del tratamiento del razonamiento sobre las acciones en el cálculo de situaciones.

La solución parcial del problema del marco ha reavivado el interés sobre la aproximación declarativa del razonamiento acerca de las acciones, que se ha visto eclipsado por los sistemas de planificación de propósito específico desde principios de los años 70 (*véase* el Capítulo 11). Bajo la etiqueta de **robótica cognitiva** se ha realizado un enorme progreso acerca de las representaciones lógicas de la acción y el tiempo. El lenguaje GOLOG usa toda la potencia expresiva de la lógica de la programación para describir acciones y planes (Levesque *et al*., 1997a) y se ha extendido para manejar acciones concurrentes (Giacomo *et al*., 2000), entornos estocásticos (Boutilier *et al*., 2002) y percepción a través de los sentidos (Reiter, 2001a).

ROBÓTICA COGNITIVA

El cálculo de eventos fue introducido por Kowalski y Sergot (1986) para manejar tiempo continuo, y han surgido numerosas variaciones (Sadri y Kowalski, 1995). Shanahan (1999) presenta una sencilla e interesante revisión. James Allen introdujo intervalos de tiempo por la misma razón (Allen, 1983, 1984), argumentando que los intervalos eran mucho más naturales que las situaciones para el razonamiento sobre eventos extendidos y concurrentes. Peter Ladkin (1986a, 1986b) introdujo intervalos temporales «cóncavos» (intervalos con huecos, esencialmente uniones de intervalos temporales «convexos») y aplicó las técnicas del álgebra abstracta matemática a la representación del tiempo. Shoham (1987) describe la reificación de eventos y conjuntos a partir de un nuevo esquema de su invención para tal propósito. Existen aspectos significativos en común entre la ontología basada en eventos vista en este capítulo y los análisis de eventos debidos al filósofo Donald Davidson (1980). Las **historias** de Pat Hayes (1985a) como ontología de líquidos también tienen mucho en común.

El tema de la ontología de las sustancias también tiene una larga historia. Plato propuso que las sustancias fueran entidades abstractas totalmente distintas a los objetos físicos. Él diría *HechoDe*(*Mantequilla*$_3$, *Mantequilla*) en lugar de *Mantequilla*$_3 \in$ *Mantequilla*. Esto lleva a una jerarquía de sustancias en la cual por ejemplo, *MantequillaSinSal* es una sustancia más específica que *Mantequilla*. La posición adoptada en este capítulo, en la cual las sustancias son categorías de objetos, fue defendida por Richard Montague (1973). También ha sido adoptada en el proyecto CYC. Copeland (1993) creó un serio, pero no insuperable, ataque. La aproximación mencionada en el capítulo, en el que la mantequilla es un objeto que consiste en todos los objetos mantecosos del universo, fue propuesto originalmente por el logístico polaco Lesniewski (1916). Su **mereología** (el nombre se deriva de la palabra griega que significa «parte») usa la relación parte-conjunto como un sustituto para la teoría de conjuntos matemáticos, con la finalidad de eliminar entidades abstractas como los conjuntos. Una exposición de estas ideas fácil de leer la dan Leonard y Goodman (1940), y el trabajo *The Structure of Appearance* (Goodman, 1977) aplica las ideas a varios problemas de representación del conocimiento. Aunque algunos aspectos de la aproximación mereológica son confusos (por ejemplo, la necesidad de un mecanismo de herencia separado basado en relaciones parte-conjunto), la aproximación ganó el apoyo de Quine (1960). Harry Bunt (1985) ha proporcionado un análisis exhaustivo de su uso en la representación del conocimiento.

MEREOLOGÍA

Los objetos mentales y los estados han sido objeto de intenso estudio en Filosofía e IA. La **lógica modal** es el método clásico para el razonamiento acerca del conocimien-

LÓGICA MODAL

to en Filosofía. La lógica modal añade operadores modales a la lógica de primer orden, tales como *B* (cree) y *K* (conoce), que reciben *sentencias* en lugar de términos como argumentos. La teoría de la demostración en la lógica modal limita la sustitución al ámbito de los contextos modales, lo que le permite lograr opacidad referencial. La lógica modal del conocimiento fue inventada por Jaakko Hintikka (1962). Saul Kripke (1963) definió la semántica de la lógica modal del conocimiento en función de **mundos posibles**. En términos generales, se dice que un mundo es posible para un agente siempre y cuando dicho mundo sea congruente con todo lo que el agente sabe. Con base en lo anteiror, se pueden deducir reglas de inferencia en las que participe el operador *K*. Robert C. Moore vincula la lógica modal del conocimiento con un estilo para razonar sobre el conocimiento que hace referencia directa a los mundos posibles de la lógica de primer orden (Moore, 1980, 1985). Si bien la lógica modal puede parecer a veces un campo arcano e intimidante, son muchas sus aplicaciones en el razonamiento acerca de la información en sistemas distribuidos. El libro *Reasoning about Knowledge* de Fagin *et al*. (1995) proporciona una esmerada introducción a la aproximación modal. La conferencia bianual *Theoretical Aspects of Reasoning About Knowledge* (TARK) abarca aplicaciones de la teoría del conocimiento en IA, ciencias económicas y sistemas distribuidos.

La teoría sintáctica de los objetos mentales fue por primera vez estudiada en profundidad por Kaplan y Montague (1960), quienes mostraron que se puede llegar a paradojas si no se maneja con cuidado. Debido a que tiene un modelo natural en términos de creencias como configuraciones físicas de un computador o un cerebro, se ha hecho popular en IA en los últimos años. Konolige (1982) y Haas (1986) la usaron para describir motores de inferencia de potencia limitada, mientras que Morgenstern (1987) mostró cómo podría ser usada para describir precondiciones del conocimiento en planificación. Los métodos para acciones basadas en la observación de planes descritos en el Capítulo 12 están basados en la teoría sintáctica. Ernie Davis (1990) establece una excelente comparación entre las teorías del conocimiento sintáctica y modal.

El filósofo griego Porphyry (234-305 a.C.) comenta sobre las *Categorías* de Aristóteles que establecen lo que podría calificarse como la primera red semántica. Charles S. Peirce (1909) desarrolló grafos existenciales como el primer formalismo de redes semánticas usando lógica moderna. Ross Quillian (1961), conducido por un interés acerca de la memoria humana y el procesamiento del lenguaje, inició el trabajo con las redes semánticas dentro del campo de la IA. Un artículo influyente de Marvin Minsky (1975) presentó una versión de las redes semánticas denominadas **marcos**. Un marco era una representación de un objeto o categoría, con atributos y relaciones con otros objetos o categorías. Aunque el artículo sirvió para iniciar el interés en el campo de la representación del conocimiento en sí, fue criticado por ser un conjunto de ideas recicladas desarrolladas en el campo de la programación orientada a objetos, como la herencia y el uso de valores por defecto (Dahl *et al*., 1970; Birtwistle *et al*., 1973). No está muy claro hasta qué punto los artículos posteriores sobre programación orientada a objetos fueron influenciados por trabajos anteriores de IA en el campo de las redes semánticas.

La cuestión semántica surgió muy perspicazmente en relación con las redes semánticas de Quillian (y todos aquellos que siguieron su aproximación), con su omnipresente y muy vago «enlace ES-UN», así como otros formalismos de representación del conocimiento anteriores como el de MERLIN (Moore y Newell, 1973), con sus misterio-

sas operaciones «flat» y «cover». El famoso artículo *What's In a Link?* de Wood (1975) atrajo la atención de los investigadores de IA hacia la necesidad de disponer de una semántica precisa en los formalismos de representación del conocimiento. Brachman (1979) entró en detalles sobre este punto y proporcionó soluciones. El trabajo de Patrick Hayes (1979) *The Logic of Frames* fue aún más allá, reivindicando que «la mayoría de los 'marcos' no son más que una nueva sintaxis para partes de la lógica de primer orden». Drew McDermott (1978b) en su trabajo *Tarskian Semantics, or, No Notation without Denotation!* argumentó que la aproximación basada en la teoría de modelos para la semántica usada en la lógica de primer orden debería ser aplicada a todos los formalismos para la representación del conocimiento. Esto permanece como una idea contradictoria. En particular, McDermott ha cambiado su opinión en *A Critique of Pure Reason* (McDermott, 1987). NETL (Fahlman, 1979) fue un sistema sofisticado de red semántica cuyos enlaces ES-UN (llamados «copias virtuales», o enlaces VC), se basaron más en la noción de «herencia» característica de los sistemas de marco o de los lenguajes de programación orientada a objetos, que en el subconjunto de relaciones, y fueron mucho mejor definidos que los enlaces de Quillian de la era anterior a Wood. NETL es particularmente interesante porque fue diseñado para ser implementado en *hardware* paralelo para superponerse a la dificultad de recuperar información de redes semánticas grandes. David Touretzky (1986) trata la herencia en el análisis matemático riguroso. Selman y Levesque (1993) tratan la complejidad de la herencia con excepciones, mostrando que en la mayoría de las implementaciones en NP-complejo.

El desarrollo de la lógica descriptiva es la etapa más reciente de una larga línea de investigación que ha tenido como objetivo el encontrar subconjuntos de la lógica de primer orden, para los cuales la inferencia sea computacionalmente tratable. Hector Levesque y Ron Brachman (1987) mostraron que ciertas construcciones lógicas (en particular, ciertos usos de la disyunción y la negación) fueron responsables en primer grado de que la herencia en lógica fuera no tratable computacionalmente. Basados en el sistema KL-ONE (Schmolze y Lipkis, 1983), se han desarrollado un gran número de sistemas cuyos diseños incorporan los resultados del análisis teórico de la complejidad, destacando KRYPTON (Brachman *et al*., 1983) y CLASSIC (Borgida *et al*., 1989). El resultado ha sido un incremento notable en la velocidad de la inferencia y un mejor entendimiento de la interacción entre complejidad y expresividad en los sistemas de razonamiento. Calvanese *et al*. (1999) realiza un resumen del estado del arte. En contra de esta tendencia Doyle y Patil (1991) han argumentado que restringir la expresividad de un lenguaje hace imposible resolver ciertos tipos de problemas o fomenta al usuario a rodear las restricciones del lenguaje utilizando significados no lógicos.

Los tres formalismos principales para tratar con la inferencia no monotónica (circunscripción (McCarthy, 1980), lógica por defecto (Reiter, 1980) y lógica no monotónica modal (McDermott y Doyle, 1980)), fueron introducidos en un volumen especial de la revista de IA. La programación de conjunto de respuestas puede ser vista como una extensión de la negación como fallo, o como un refinamiento de la circunscripción. La teoría de base de la semántica estable de modelos fue introducida por Gelfond y Lifschitz (1988), y los sistemas de referencia para la programación de conjunto de respuestas son DLV (Eiter *et al*., 1998) y SMODELS (Niemelä *et al*., 2000). El ejemplo de la unidad de disco procede del manual de usuario de SMODELS (Syrjänen, 2000). Lifschitz (2001)

aborda el tema de la programación de conjunto de respuestas aplicado a planificación. Brewka *et al*. (1997) hace una buena revisión de varias aproximaciones a la lógica no monotónica. Clark (1978) trata la aproximación de negación como fallo en la programación lógica y la completitud de Clark. Van Emden y Kowalski (1976) muestran que cualquier programa Prolog sin la negación tiene un modelo mínimo único. Últimamente se ha renovado el interés en la aplicación de lógica no monotónica a los sistemas de representación de conocimiento a gran escala. El sistema BENINQ para el manejo de preguntas sobre los beneficios de seguros seguramente fue la primera aplicación comercial con éxito de un sistema de herencia no monotónica (Morgenstern, 1998). Lifschitz (2001) trata sobre la aplicación de la programación de conjunto de respuestas aplicado a planificación. Un amplio conjunto de sistemas de razonamiento no monotónicos basados en programación lógica se encuentran documentados en las actas de la conferencia en *Logic Programming and Nonmonotonic Reasoning* (LPNMR).

El estudio de sistemas de mantenimiento de verdad comenzó con los sistemas TMS (Doyle, 1979) y RUP (McAllester, 1980), los cuales eran esencialmente un SMVJ. La aproximación de los SMVS se describe en una serie de artículos de Johan de Kleer (1986a, 1986b, 1986c). *Building Problem Solvers* (Forbus y de Kleer, 1993) explica en profundidad cómo se pueden utilizar los SMV en aplicaciones de IA. Nayak y Williams (1997) muestran cómo un SMV eficiente hace posible planear las operaciones de una nave espacial de la NASA en tiempo real.

Por razones obvias este capítulo no cubre *todas* las áreas de la representación del conocimiento en profundidad. Los temas principales omitidos son los siguientes:

FÍSICA CUALITATIVA

- **Física cualitativa**: subcampo de la representación del conocimiento que se ocupa específicamente de la construcción de una teoría lógica, no numérica de los objetos y procesos físicos. El término fue acuñado por Johan de Kleer (1975), si bien podría considerarse a Fahlman (1974) como el precursor con su planificador BUILD. Éste era un planificador complejo para construir complicadas torres de bloques. Durante el proceso de diseño, Fahlman se dio cuenta de que la mayor parte del trabajo (80 por ciento) se invertía en modelar la física del mundo de los bloques para determinar la estabilidad de diversos subconjuntos de bloques, en vez de hacerlo dentro de la planificación en sí. Esbozó un hipotético procedimiento semejante a la física intuitiva (*naive*) para explicar el por qué los niños son capaces de resolver problemas semejantes a los de BUILD sin recurrir a la veloz aritmética de punto flotante empleada en el modelado físico de BUILD. Hayes (1985a) emplea «historias», segmentos espacio-temporales de cuatro dimensiones semejantes a los eventos de Davidson, para construir una física intuitiva de los líquidos bastante compleja. Hayes fue el primero en demostrar que una bañera con un tapón terminará desbordándose si el grifo se mantiene abierto y que una persona que cae en un lago terminará empapada. De Kleer y Brown (1985) y Ken Forbus (1985) intentaron la construcción de algo así como una teoría de propósito general para el mundo físico, basada en las abstracciones cualitativas de ecuaciones físicas. En los últimos años, la física cualitativa se ha desarrollado al grado de que permite el análisis de una impresionante diversidad de sistemas físicos complejos (Sacks y Joskowicz, 1993; Yip, 1991). Las técnicas cualitativas se han empleado también para construir novedosos diseños de relojes, lim-

piaparabrisas y robots de seis piernas (Subramanian, 1993; Subramanian y Wang, 1994). Una buena introducción a este campo es la colección *Readings in Qualitative Reasoning about Physical Systems* (Weld y de Kleer, 1990).

RAZONAMIENTO ESPACIAL

- **Razonamiento espacial**: el razonamiento necesario para navegar en el mundo de *wumpus* y el del mercado es trivial comparado con la rica estructura espacial del mundo real. El esfuerzo más completo por capturar el razonamiento con sentido común acerca del espacio está plasmado en el trabajo de Ernest Davis (1986; 1990). El cálculo de conexión de regiones de Cohn *et al*., (1997) utiliza una forma de razonamiento espacial cualitativo y ha llevado a nuevas clases de sistemas de información geográfica. Como en el caso de la física cualitativa, muestra que los agentes pueden avanzar bastante, por decirlo así, sin recurrir a la ayuda de la representación métrica. Cuando tal representación es necesaria, se pueden emplear las técnicas diseñadas por la robótica (Capítulo 25).

RAZONAMIENTO PSICOLÓGICO

- **Razonamiento psicológico**: consiste en el desarrollo de una *psicología* de trabajo para que los agentes la utilicen al razonar sobre sí mismos y otros agentes. A menudo, está basada en lo que se conoce como psicología popular; aquellas teorías que en general utilizan los humanos al razonar sobre sí mismos y sobre otros seres humanos. Cuando los investigadores de IA proporcionan a sus agentes artificiales teorías psicológicas para razonar sobre otros agentes, tales teorías por lo general se basan en la descripción de los investigadores del propio diseño de los agentes lógicos. El razonamiento psicológico es realmente más útil en el contexto de la interpretación del lenguaje natural, en donde la capacidad para anticipar las intenciones del hablante es de suma importancia.

Las actas de las conferencias internacionales sobre los *Principios de la Representación del Conocimiento y del Razonamiento* constituyen las fuentes más actualizadas de los trabajos que se realizan en esta área. *Readings in Knowledge Representation* (Brachman y Levesque, 1985) y *Formal Theories of the Commonsense World* (Hobbs y Moore, 1985) son excelentes antologías de la representación del conocimiento. La primera se enfoca más en artículos de importancia histórica, en lenguajes de representación y formalismos, mientras que la segunda lo hace en el conocimiento acumulado en este campo. Davis (1990), Stefik (1995) y Sowa (1999) proporcionan introducciones a los libros de texto acerca de la representación del conocimiento.

EJERCICIOS

10.1 Escriba sentencias para definir los efectos de la acción *Disparar* en el mundo de *wumpus*. Describa sus efectos en *wumpus* y recuerde que el disparo usa la flecha del agente.

10.2 En el cálculo de situaciones, escriba un axioma para asociar el tiempo 0 con la situación S_0 y otro axioma para asociar el tiempo t con cualquier situación derivada de S_0 a través de una secuencia t de acciones.

10.3 En este ejercicio, se considerará el problema de planear una ruta para un robot para ir desde una ciudad a otra. La acción básica que puede tomar el robot es *Ir*(*x*, *y*), que lo lleva desde la ciudad *x* a la ciudad *y* si existe una ruta directa entre esas ciudades. *RutaDirecta*(*x*, *y*) es cierto si y sólo si, existe una ruta directa entre *x* e *y*. Se puede asumir que todos estos hechos están ya en la base de conocimiento (*véase* el mapa en el Apartado 3.1). El robot comienza en Arad y debe llegar a Bucarest.

a) Escriba una descripción lógica adecuada para la situación inicial del robot.
b) Escriba una petición lógica adecuada cuyas soluciones proporcionarán rutas posibles hacia el objetivo.
c) Escriba una sentencia describiendo al acción *Ir*.
d) Ahora supóngase que siguiendo la ruta directa entre dos ciudades se consume una cantidad de combustible igual a la distancia entre las ciudades. El robot comienza con el depósito lleno de combustible. Aumente la representación para incluir estas consideraciones. La descripción de la acción debe ser tal, que la petición formulada anteriormente proporcione planes viables.
e) Describa la situación inicial, y escriba una nueva regla o conjunto de reglas para describir la acción *Ir*.
f) Ahora supóngase que algunos de los vértices tienen también gasolineras, donde el robot puede llenar su depósito. Extienda la representación y escriba todas las reglas necesarias para describir las gasolineras, incluyendo la acción de *Repostar*.

10.4 Utilizando los axiomas de la Sección 10.3, muestre los pasos de razonamiento lógico que permiten concluir que la acción *Ambos*(*Impulsar*(*Izquierda*), *Impulsar*(*Derecha*)) no derramarán la sopa.

10.5 Represente las siguientes siete sentencias utilizando y extendiendo las representaciones desarrolladas en el capítulo:

a) El agua es líquida ente 0 y 100 grados.
b) El agua hierve a 100 grados.
c) El agua de la botella de John está congelada.
d) Perrier es un tipo de agua.
e) John tiene agua de tipo Perrier en su botella.
f) Todos los líquidos tienen un punto de congelación.
g) Un litro de agua pesa más que un litro de alcohol.

Ahora repita el ejercicio usando una representación que utilice mereología, en el cual, por ejemplo, *Agua* es un objeto que contiene como partes toda el agua del mundo.

10.6 Escriba definiciones para lo siguiente:

a) *DescomposiciónDeParteExhaustiva*.
b) *PparticiónDeParte*.
c) *PartesDisjuntas*.

Esto debería ser análogo a la definición de *DecomposiciónExhaustiva*, *Partición* y *Disyunción*. ¿Es el caso de *PparticiónDeParte*(*s*, *GrupoDe*(*s*))?, si es así demuéstrelo, si no, proporcione un contraejemplo y defina las condiciones suficientes bajo las cuales sí es cierto.

10.7 Escriba un conjunto de sentencias que permitan calcular el precio de un tomate individual (u otro objeto), dado el precio por kilo. Extienda la teoría para permitir el cálculo del precio de una bolsa de tomates.

10.8 Un esquema alternativo para representar medidas consiste en aplicar la función de medida a un objeto de longitud abstracta. En este esquema, ¿se podría escribir *Pulgadas*(*Longitud*(L_1)) = 1,5?, ¿cómo se compara este esquema con el visto en el capítulo? Algunos aspectos a tener en cuenta son la necesidad de incluir axiomas de conversión, nombres para cantidades abstractas (como «50 dólares») y comparaciones de medidas abstractas en diferentes unidades (50 pulgadas son más que 50 centímetros).

10.9 Construya una aplicación para intercambio de monedas que permita fluctuaciones diarias del cambio de moneda.

10.10 Este ejercicio contempla las relaciones entre categorías de eventos y el intervalo de tiempo en el cual ocurren:

a) Defina el predicado $T(c, i)$ en términos de *SubEvento* y $\in$.
b) Explique de forma precisa por qué no se necesitan dos notaciones diferentes para describir categorías de eventos disjuntas.
c) Dé una definición formal para $T(UnoDe(p, q), i)$ y $T(O(p, q), i)$.
d) Explique si tiene sentido tener dos formas de negación de eventos, análogas a las dos formas de disyunción. Denomine *No* y *Nunca* y dé para ellas una definición formal.

10.11 Defina el predicado *Constante*, donde *Constante*(*Localización*(x)) significa que la localización del objeto x no cambia en el tiempo.

10.12 Defina los predicados *Antes*, *Después*, *Durante* y *Superpuesto*, utilizando el predicado *Coincidir* y las funciones *Inicio* y *Fin*, pero no la función *Tiempo* o el predicado $<$.

10.13 En la Sección 10.5 se utilizaron los predicados *Enlace* y *TextoDeEnlace* para describir conexiones entre páginas web. Utilice los predicados *EnEtiqueta* y *ObtenerPágina*, entre otros, para escribir definiciones para *Enlace* y *TextoDeEnlace*.

10.14 Una parte del proceso de compra que no fue cubierto en este capítulo es la comprobación de compatibilidad entre artículos. Por ejemplo, si un cliente pide un computador, ¿se conecta adecuadamente con los periféricos deseados? Si se solicita una cámara digital, ¿tiene la tarjeta de memoria correcta y las baterías? Escriba una base de conocimiento que decida cuándo un conjunto de artículos es compatible y puede ser utilizado para sugerir cambios o artículos adicionales si los originales no son compatibles. Asegúrese de que la base de conocimiento trabaja con al menos una gama de productos y es fácilmente extensible a otras.

10.15 Añada reglas para extender la definición del predicado *Nombre*(s, c) de tal forma que una cadena como «computador portátil» se corresponda con los nombres de categorías adecuados de diferentes almacenes. Trate de hacer la definición general. Compruebe si funciona buscando en diez almacenes *online* y en los nombres de categorías que tiene cada uno. Por ejemplo, para la categoría de portátiles, se encuentran los nom-

bres «Notebooks», «Portátiles», «Computadores notebook», «Portátiles y notebooks» y «Notebooks PC». Algunos de ellos pueden ser soportados por ocurrencias *Nombre* explícitas, mientras que otros podrían ser implementados por reglas para manejar plurales, conjunciones, etc.

10.16 Una solución completa al problema de las correspondencias inexactas en las descripciones de un cliente respecto a su compra es muy difícil, y requiere un completo despliegue de procesamiento de lenguaje natural y técnicas de recuperación de información (*véanse* los Capítulos 22 y 23). Un pequeño paso es permitir al usuario especificar unos valores mínimos y máximos para varios atributos. Se insistirá en que el usuario utilice la siguiente gramática a la hora de describir sus productos:

$$
\begin{array}{lll}
\textit{Descripción} & \rightarrow & \textit{Categoría}\,[\textit{Conector Modificador}]* \\
\textit{Conector} & \rightarrow & \text{«}\textit{con}\text{»} \mid \text{«}\textit{y}\text{»} \mid \text{«,»} \\
\textit{Modificador} & \rightarrow & \textit{Atributo} \mid \textit{Atributo Op Valor} \\
\textit{Op} & \rightarrow & \text{«=»} \mid \text{«>»} \mid \text{«<»}
\end{array}
$$

Aquí, *Categoría* da nombre a una categoría de productos, *Atributo* es cualquier característica como «CPU» o «precio» y *Valor* es el valor objetivo para el atributo. Por lo tanto, la petición «computador con al menos una CPU de 2,5 GHz por menos de 1.000 dólares» se ha de expresar como «computador con CPU $>$ 2,5 GHz y precio $<$ 1.000 dólares». Implemente un agente de compra que acepte descripciones en este lenguaje.

10.17 En la descripción de compra por Internet se omitió el paso importante de realmente *comprar* el producto. Proporcione una descripción lógica formal para comprar, utilizando cálculo de eventos. Es decir, defina la secuencia de eventos que ocurren cuando un comprador envía su pedido con el número de su tarjeta y finalmente se emite un recibo y recibe el producto.

10.18 Describa el evento de intercambiar algo por otra cosa. Describa la compra como una clase de intercambio donde uno de los objetos intercambiados es una cantidad de dinero.

10.19 Los dos ejercicios anteriores asumen una noción bastante primitiva de la propiedad. Por ejemplo, el comprador comienza *poseyendo* los billetes. Este panorama comienza a desmoronarse cuando, por ejemplo, el dinero de uno está en el banco, porque no hay ninguna colección específica de billetes que uno posee. El panorama se complica cuando se utilizan préstamos, *leasing*, alquileres y fianzas. Investigue los aspectos de sentido común y conceptos legales sobre la pertenencia, y proponga un esquema mediante el cual puedan ser representados formalmente.

10.20 Se trata de crear un sistema para aconsejar a los alumnos de informática sobre qué cursos seleccionar durante un período de tiempo prolongado para satisfacer los requerimientos del programa (use cualquier requerimiento que sea adecuado para su institución). Primero, decida un vocabulario para representar toda la información y después represéntela. A continuación utilice una petición adecuada para el sistema, que devolverá un programa legal de estudios como solución. Se debería permitir, para adaptar el sistema a usuarios individuales, que el sistema preguntara por los cursos o equivalencias que los estudiantes ya han cursado, y no generar programas que repitan dichos cursos.

Sugiera formas en las que el sistema podría ser mejorado (por ejemplo teniendo en cuenta el conocimiento sobre las preferencias de los estudiantes, la carga de trabajo, instructores buenos y malos etc.). Para cada clase de conocimiento, explique cómo podría ser expresado lógicamente. ¿Podría el sistema incorporar fácilmente esta información para encontrar el *mejor* programa de estudios para un estudiante?

10.21 La Figura 10.1 muestra los niveles superiores de una jerarquía para cualquier cosa. Extiéndase para incluir tantas categorías como sean posibles. Una buena forma de hacer esto es cubrir todas las cosas de la vida a diario. Esto incluye objetos y eventos. Comience por levantarse y seguir de una forma ordenada anotando todo lo que uno ve, toca, hace y piensa. Por ejemplo, un ejemplo de prueba aleatoria produce música, noticias, leche, andar, conducir, repostar, soda, alfombra, hablar, profesor Fateman, pollo al curry, lengua, 7 dólares, sol, la prensa del día, etc.

Se debería generar una gráfico de jerarquía simple (en un trozo de papel grande) y un listado de objetos y categorías con las relaciones que satisfacen los miembros de cada categoría. Todo objeto debería estar en una categoría, y cada categoría debería estar en la jerarquía.

10.22 (Adaptado de un ejemplo de Doug Lenat.) La misión es capturar, de una forma lógica, conocimiento suficiente para responder una serie de preguntas sobre la siguiente sentencia simple:

> Ayer John fue al supermercado Safeway del norte de Berkeley y compró dos libras de tomates y una libra de carne picada.

Comience por tratar de representar el contenido de la sentencia en un conjunto de aserciones. Se deberían escribir sentencias con una estructura lógica sencilla (por ejemplo, sentencias en las que los objetos tienen ciertas propiedades, en las que los objetos se relacionan de cierta forma, en las que los objetos que satisfacen una propiedad satisfacen otra). Lo siguiente puede ayudar para empezar:

- ¿Qué clases, objetos y relaciones son necesarios? ¿Cuáles son sus padres, hermanos y demás? (Se necesitarán eventos y ordenamiento temporal, entre otras cosas.)
- ¿En qué lugar tendrían cabida en una jerarquía más general?
- ¿Cuáles son las restricciones y las interrelaciones entre ellos?
- ¿Qué nivel de detalle se debe tener con cada uno de los conceptos?

La base de conocimiento que se construya deber ser capaz de responder una lista de preguntas que se proporcionarán en breve. Algunas de las preguntas tratan sobre el material dado explícitamente en el texto, pero la mayoría requieren tener otro conocimiento de fondo (una lectura entre líneas). Se deberá tratar con las clases de cosas que hay en un supermercado, lo que rodea el proceso de comprar las cosas seleccionadas, para qué serán usadas, etc. Trate de realizar la representación lo más general posible. Por dar un ejemplo trivial: no diga «La gente compra comida en Safeway», porque esto no le ayudará con la gente que compra en otro supermercado. No diga «Joe hace spaghetti con tomate y carne picada», porque no le ayudará con absolutamente nada más. Además, no le dé la vuelta a las preguntas convirtiéndolas en respuestas. Por ejemplo, la pregunta (c) dice: «¿Compró John carne?» (no «¿Compró John una libra de carne picada?»).

Esboce el proceso de razonamiento que respondería a las preguntas. Para realizarlo, necesitará sin lugar a dudas crear conceptos adicionales, realizar nuevas aserciones, etc. Si es posible, utilice un sistema de razonamiento lógico para demostrar la suficiencia de su base de conocimiento. La mayoría de las cosas que escriba no serán del todo correctas en la realidad, pero no se preocupe demasiado. La idea es extraer el sentido común que le permita responder a todas las cuestiones. Una respuesta totalmente correcta a todas las preguntas es *extremadamente* compleja, probablemente más allá del estado de arte en las investigaciones actuales sobre representación del conocimiento. Pero usted debería ser capaz de proporcionar un conjunto consistente de axiomas para las preguntas en concreto aquí expresadas.

a) ¿John es un niño o un adulto? [Adulto]
b) ¿Tiene John ahora por lo menos dos tomates? [Sí]
c) ¿Compró John carne? [Sí]
d) ¿Si Mary estaba comprando tomates al mismo tiempo que John, le vio él a ella? [Sí]
e) ¿Se hacen los tomates en el supermercado? [No]
f) ¿Qué va a hacer John con los tomates? [Comérselos]
g) ¿Se vende desodorante en Safeway? [Sí]
h) ¿Trajo John dinero al supermercado? [Sí]
i) ¿Tiene John menos dinero después de ir al supermercado? [Sí]

10.23 Añada o realice los cambios necesarios a la base de conocimiento del ejercicio anterior para que las preguntas que siguen a continuación se puedan responder. Muestre que se pueden realmente responder utilizando la base de conocimiento, e incluya en el informe una referencia de los cambios, explicando por qué fueron necesarios, si fueron grandes o pequeños, etc.

a) ¿Hay otra gente en Safeway mientras John está allí? [Sí, ¡el personal!]
b) ¿Es John vegetariano? [No]
c) ¿De quién es el desodorante de Safeway? [De la compañía Safeway]
d) ¿Obtuvo John una libra de carne picada? [Sí]
e) ¿Tenía la gasolinera cerca de su puerta combustible? [Sí]
f) ¿Cupieron los tomates en el maletero del coche de John? [Sí]

10.24 Recuerde que la información sobre la herencia en las redes semánticas puede ser capturada lógicamente por sentencias de implicación adecuadas. En este ejercicio, se considera la eficiencia de usar estas sentencias para la herencia.

a) Considere la información contenida en un catálogo de coches usados como el Libro Azul de Kelly (por ejemplo, que las furgonetas Dodge de 1973 valen 575 dólares). Suponga que toda esta información (para 11.000 modelos) está codificada en reglas lógicas como se sugiere en este capítulo. Escriba tres reglas así, incluyendo las de la furgoneta Dodge de 1973. ¿Cómo usaría las reglas para encontrar el valor de un vehículo concreto (por ejemplo, JB), el cual es una furgoneta Dodge de 1973?
b) Compare la eficiencia en tiempo del método de encadenamiento hacia atrás para resolver este problema con el método de herencia usado en las redes semánticas.

c) Explique cómo el encadenamiento hacia delante permite a un sistema basado en lógica resolver el mismo problema eficientemente, suponiendo que la base de conocimiento sólo contiene las 11.000 reglas sobre precios.

d) Describa una situación en la que ni el encadenamiento hacia delante ni el que va hacia atrás permitirá manejar eficientemente una consulta de precio para un vehículo particular.

e) ¿Puede sugerir una solución que permita resolver eficientemente este tipo de consulta en todos los casos en sistemas lógicos? (*Pista:* Recuerde que dos vehículos de la misma categoría tienen el mismo precio.)

10.25 Uno podría suponer que la distinción sintáctica entre los enlaces no encuadrados y los encuadrados de forma simple en las redes semánticas es innecesaria, porque los enlaces encuadrados de forma simple están unidos siempre a categorías. Un algoritmo de herencia podría asumir simplemente que un enlace no encuadrado debe aplicarse a todos los miembros de esa categoría. Muestre que este argumento es falaz, dando ejemplos de los errores que podrían surgir.

11 Planificación

Donde veremos cómo un agente puede extraer ventaja del conocimiento de la estructura de un problema para construir complejos planes de acción.

Llamaremos **planificación** al proceso de búsqueda y articulación de una secuencia de acciones que permitan alcanzar un objetivo. Hemos visto dos ejemplos de agentes planificadores hasta el momento: el agente solucionador de problemas basado en búsquedas, Capítulo 3, y el agente planificador lógico del Capítulo 10. Este capítulo se ocupará principalmente *de ampliar* el estudio a problemas de planificación complejos que no pueden abordarse mediante los enfoques propuestos hasta ahora.

La Sección 11.1 presenta un lenguaje adecuado para la formulación de problemas de planificación, incluyendo acciones y estados. Este lenguaje está estrechamente relacionado con el que se mostró para la representación proposicional y de primer-orden de acciones, estudiadas en los Capítulos 7 y 10. La Sección 11.2 muestra cómo los algoritmos de búsqueda hacia-adelante y hacia-atrás aprovechan esta forma de representación, principalmente a través de heurísticas adecuadas que pueden derivarse automáticamente de la estructura de la representación de los problemas (análogamente al modo en que vimos en el Capítulo 5 cómo se construyeron heurísticas adecuadas en problemas a los que se imponían restricciones). La Secciones 11.3, 11.4 y 11.5 describen algoritmos de planificación que van más allá que las búsquedas hacia-adelante y hacia-atrás, aprovechándose de la información proporcionada por la estructura de los problemas. En particular, exploraremos enfoques que no están restringidos exclusivamente a la consideración de secuencias de acciones ordenadas.

En este capítulo, tendremos en cuenta entornos que son completamente observables, deterministas, finitos, estáticos (los cambios suceden sólo cuando los agentes actúan) y discretos (en tiempo, acciones, objetos y efectos). Estos contextos se deno-

PLANIFICACIÓN CLÁSICA

minan entornos de **planificación clásica.** En contraste, la planificación no-clásica se ocupará de contextos parcialmente observables o estocásticos donde se aplicarán diferentes propuestas y diseños de agentes y algoritmos, tal cómo se expondrá en los Capítulos 12 y 17.

11.1 El problema de planificación

Consideremos qué sucedería si un agente solucionador de problemas, que utilizase algoritmos de búsqueda clásicos para llevar a cabo su tarea (búsqueda en profundidad, A*, etc.), se enfrentase a problemas en entornos reales. Este planteamiento nos ayudará a diseñar mejores agentes de planificación.

La primera dificultad y más obvia es la posibilidad de que el agente pueda ser desbordado por acciones irrelevantes al problema. Consideremos la tarea de comprar un ejemplar del libro «*Inteligencia Artificial: un enfoque moderno*» en una librería *online*. Supongamos que tenemos una acción de compra para cada número ISBN de 10 dígitos y para un total de 10 billones de acciones. El algoritmo de búsqueda tendría que examinar los resultados de los 10 billones de acciones para encontrar una que satisficiese el objetivo de adquirir la copia de un libro con ISBN 0137903952. En contraposición, un agente de planificación razonable debería ser capaz de trabajar con expresiones de objetivos explícitas tales como *Tener*(*ISBN 0137903952*) y generar la acción *Comprar*(*ISBN 0137903952*) directamente. Para hacer esto, el agente simplemente necesita un conocimiento general del tipo «*Comprar*(x) se reduce a *Tener*(x)». Dado este conocimiento y el objetivo propuesto, el planificador puede decidir en un único paso unificado que *Comprar*(*ISBN 0137903952*) es la acción correcta.

La siguiente dificultad es encontrar una **función heurística** adecuada. Supongamos que el objetivo del agente es comprar cuatro libros diferentes en una librería *online*. Tendremos 10^{40} planes formados por cuatro etapas, por lo que una búsqueda sin recurrir a la ayuda de una heurística no puede ser ni cuestionada. Es obvio que para un humano sería una buena estimación heurística que el coste de un estado fuese el número de libros que permanecen sin ser comprados; desafortunadamente, esta visión perspicaz no es obvia para un agente que evalúa su objetivo como una caja negra que devuelve un valor Verdadero o Falso para cada estado. Por tanto, al agente solucionador de problemas le falta autonomía; requiere de un humano que le suministre una función heurística para cada nuevo problema. Sin embargo si un agente planificador tiene acceso a una representación explícita del objetivo como una secuencia de subobjetivos, puede utilizar una simple heurística *independiente de dominio*: el número de conjunciones insatisfechas. Para el problema anterior de la compra del libro, el objetivo podría ser $Tener(A) \wedge Tener(B) \wedge Tener(C) \wedge Tener(D)$, y un estado conteniendo $Tener(A) \wedge Tener(C)$ podría presentar coste dos. De este modo, el agente automáticamente adquiere la heurística correcta para este problema y para otros. Veremos más tarde en el capítulo cómo construir heurísticas sofisticadas que examinen tanto las acciones disponibles como la estructura del objetivo.

DESCOMPOSICIÓN DE PROBLEMAS

Finalmente, el solucionador de problemas podría ser ineficaz porque no pudiera aprovecharse de la **descomposición del problema.** Consideremos el problema de trasladar un conjunto de maletas de viaje a sus respectivos destinos, los cuales están distribuidos a lo largo de Australia. Una estrategia lógica sería buscar el aeropuerto más próximo a cada destino y dividir el problema total en varios subproblemas, uno por cada aeropuerto. Para cada conjunto de maletas asignadas a un aeropuerto determinado, el problema se descompone más profundamente, en función del destino concreto. Vimos en el Capítulo 5 que este tipo de descomposición contribuye a la eficiencia en la resolución de problemas que deben satisfacer restricciones impuestas. Esto es igualmente válido para planificadores: en el peor de los casos, el orden temporal para encontrar el mejor plan que entregase n paquetes sería $O(n!)$, pero solamente un orden temporal $O((n/k)! \times k)$ si el problema puede ser descompuesto en k partes iguales.

PRÁCTICAMENTE DESCOMPONIBLES

Como hicimos notar en el Capítulo 5, problemas perfectamente descomponibles no son frecuentes[1]. El diseño de muchos sistemas de planificación (especialmente los planificadores de orden-parcial descritos en la Sección 11.3) están basados en la hipótesis de que la mayor parte de los problemas originados en contextos reales son **prácticamente descomponibles.** Esto es, el planificador puede trabajar siguiendo subobjetivos de manera independiente, aunque podría necesitar trabajo adicional para combinar el resultado de los subplanes generados. Para algunos problemas, esta hipótesis no es válida porque trabajar en la consecución de un subobjetivo probablemente deje de lado algún otro. Estas interacciones entre subobjetivos son uno de los motivos que hacen a los puzles desconcertantes.

El lenguaje de los problemas de planificación

La discusión precedente sugiere que la representación de los problemas de planificación (estados, acciones y objetivos) debe hacer posible que los algoritmos de planificación se aprovechen de la estructura lógica del problema. La clave está en encontrar un lenguaje que sea suficientemente expresivo para describir un amplio rango de problemas, pero suficientemente restrictivo para permitir algoritmos operativos y eficientes. En esta sección, destacaremos en primer lugar el lenguaje de representación básico de los planificadores clásicos, conocido como el lenguaje STRIPS[2]. Posteriormente, expondremos algunas de las diferentes modificaciones que se han desarrollado de lenguajes tipo-STRIPS.

Representación de estados. Los planificadores descomponen el mundo en términos de condiciones lógicas y representan un estado como una secuencia de literales positivos conectados. Consideraremos literales proposicionales: por ejemplo, $Pobre \wedge Desconocido$ puede representar el estado de un agente desafortunado. Utilizaremos también literales de primer-orden como por ejemplo, $En(Avión_1, Melbourne) \wedge En(Avión_2, Sydney)$, que

[1] Nótese que incluso la entrega de un paquete no es un problema perfectamente descomponible. Pueden existir casos en los cuales sea mejor asignar paquetes a un aeropuerto más lejano si eso evita un vuelo a un aeropuerto más cercano innecesariamente. Sin embargo, muchas compañías de transporte prefieren la simplicidad organizacional y computacional de soluciones descomponibles.

[2] STRIPS significa *STandford Research Institute Problem Solver.*

puede representar un estado en el problema del reparto de equipajes. Los literales, en un marco de descripciones de primer orden, deben ser **simples** y **sin dependencias funcionales**. Por ejemplo, literales del tipo *En*(*x, y*) o *En*(*Padre*(*Fred*), *Sydney*) no se permiten. La **hipótesis de un mundo cerrado** es asumida por la que todas las condiciones que no son mencionadas en un estado, se asume que son falsas.

Representación de objetivos. Un objetivo es un estado parcialmente especificado, representado como una secuencia de literales positivos y simples, tales como *Rico* ∧ *Famoso* o *En*(P_2, *Tahití*). Un estado proposicional *s* **satisface** un objetivo *g* si *s* contiene todos sus elementos en *g* (y posiblemente otros además). Por ejemplo, el estado *Rico* ∧ *Famoso* ∧ *Miserable* satisface el objetivo *Rico* ∧ *Famoso*.

SATISFACCIÓN DE OBJETIVOS

Representación de acciones. Una acción es especificada en términos de las precondiciones que deben cumplirse antes de ser ejecutada y de las consecuencias que se siguen cuando se ejecuta. Por ejemplo, la acción que nos indica cómo un avión vuela desde una ciudad a otra puede exponerse como:

Acción (*avión*(*p, desde, hasta*),
PRECOND:*En*(*p, desde*) ∧ *avión*(*p*) ∧ *aeropuerto*(*desde*) ∧ *aeropuerto* (*hasta*)
EFECTO:¬ *En* (*p, desde*) ∧ *En* (*p, hasta*)

ESQUEMA DE ACCIÓN

Esto es más propio llamarlo **esquema de acción**, para indicar que representa un número de diferentes acciones que pueden ser derivadas mediante la instanciación de las variables *p, desde* y *hasta* pudiendo adquirir diferentes valores. En general, un esquema de acción consta de tres partes:

- El nombre de la acción y la lista de parámetros de los que depende la acción (por ejemplo, *Volar*(*p, desde, hasta*) sirve para identificar la acción.
- La **precondición** es la unión de literales positivos sin dependencia funcional estableciendo lo que debe ser verdad en un estado antes de que una acción sea ejecutada. Todas las variables en las precondiciones deben también aparecer en la lista de parámetros de acción.
- El **efecto** es la unión de literales sin dependencia funcional describiendo cómo el estado cambia cuando la acción es ejecutada. Un literal positivo *P* en el efecto se espera que sea verdadero en el estado resultante de la acción, mientras que un literal negativo ¬*P* se espera que sea falso. Las variables en el efecto deben pertenecer a la lista de parámetros de acción.

PRECONDICIÓN

EFECTO

Para hacer más sencilla la legibilidad, algunos sistemas de planificación dividen el efecto en dos listas de literales: los positivos en la **lista Añadir**, y los negativos en la **lista Borrar.**

LISTA AÑADIR

LISTA BORRAR

Hasta ahora hemos definido la sintaxis en la representación de problemas de planificación; veremos ahora cómo definir la semántica. El modo más sencillo es mediante la descripción de cómo las acciones afectan a los estados (un método alternativo consiste en especificar una traslación directa a un conjunto de axiomas de estado sucesivo, cuya semántica viene directamente de las reglas de la lógica de primer orden; *véase* Ejercicio 11.3). Diremos que una acción es **aplicable** en cualquier estado que satisfaga sus precondiciones; en otro caso, la acción no tendrá efecto. La aplicación, para un esque-

APLICABLE

ma de acción de primer orden, se reduce a la sustitución θ de las variables en la precondición. Por ejemplo, supongamos que el estado actual está descrito por

$$En(P_1, JFK) \wedge En(P_2, SFO) \wedge Avión(P_1) \wedge Avión(P_2)$$
$$\wedge\ Aeropuerto(JFK) \wedge Aeropuerto(SFO)$$

Este estado satisface la precondición

$$En(p, desde) \wedge Avión(p) \wedge Aeropuerto(desde) \wedge Aeropuerto\ (hasta)$$

con la sustitución $\{p/P_1, desde/JFK, hasta/SFO\}$ (junto a otros; *véase* Ejercicio 11.2). De esta forma, la acción concreta $Volar(P_1, JFK, SFO)$ es aplicable.

RESULTADO

Comenzando en el estado s, el **resultado** de ejecutar una acción aplicable a nos lleva a otro estado s', que es el mismo que s excepto que cualquier literal positivo P en el efecto de a es añadido a s', y cualquier literal negativo $\neg P$ es eliminado de s'. De este modo, después de $Volar(P_1, JFK, SFO)$, el estado actual se convierte en

$$En(P_1, SFO) \wedge En(P_2, SFO) \wedge Avión(P_1) \wedge Avión(P_2)$$
$$\wedge\ Aeropuerto(JFK) \wedge Aeropuerto(SFO)$$

Notemos que si un efecto positivo está ya en s no se añade dos veces, y si un efecto negativo no está en s, entonces esa parte del efecto es ignorada. Esta definición expresa la llamada **hipótesis** STRIPS: cada literal no mencionado en el efecto permanece sin modificar. De este modo, STRIPS evita el **problema del marco** representacional descrito en el Capítulo 10.

HIPÓTESIS STRIPS

SOLUCIÓN

Finalmente, definimos la **solución** de un problema de planificación. En su forma más sencilla, es simplemente una secuencia de acciones que, ejecutada en el estado inicial, da como resultado un estado final que satisface el objetivo. Posteriormente, en este mismo capítulo, describiremos las soluciones como conjuntos de acciones parcialmente ordenados, siempre que cada secuencia de acciones que respete el orden parcial sea solución.

Expresividad y extensiones

Las diferentes restricciones impuestas por la representación STRIPS fueron elegidas con el deseo de diseñar algoritmos más simples y más eficientes, sin complicarlo demasiado para poder describir problemas reales. Una de las restricciones más importantes impuestas es que los literales no tengan *dependencia funcional* de otros atributos. Con esta restricción, estamos seguros de que dado un problema, todo sistema de acción puede ser proposicionalizado, esto es, transformado en una colección finita de representaciones de acciones estrictamente proposicionales (*véase* Capítulo 9). Por ejemplo, en el dominio del transporte aéreo para un problema de 10 aviones y cinco aeropuertos, podemos convertir la sentencia $Volar(p, desde, hacia)$ en $10 \times 5 \times 5 = 250$ acciones puramente proposicionales. Los planificadores en los Apartados 11.4 y 11.5 trabajan directamente con descripciones proposicionalizadas. Si admitimos símbolos funcionales, podremos construir infinitamente muchos más estados y acciones.

En los últimos años, se ha mostrado claramente que STRIPS no posee expresividad suficiente para ciertos dominios reales. Como resultado, muchos lenguajes han sido

ADL

desarrollados. La Figura 11.1 describe brevemente uno de los más importantes, el Lenguaje de Descripción de Acciones (*Action Description Language*, **ADL**), y la comparación con el lenguaje STRIPS. En ADL, la acción *Volar* podría ser escrita como

$$\begin{array}{l} \textit{Acción}\ (\textit{Volar}(p : \textit{Avión}, \textit{desde} : \textit{Aeropuerto}, \textit{hasta} : \textit{Aeropuerto}), \\ \quad \text{PRECOND}{:}\textit{En}\ (p, \textit{desde}) \wedge (\textit{desde} \neq \textit{hasta}) \\ \quad \text{EFECTO}{:}\neg\textit{En}\ (p, \textit{desde}) \wedge \textit{en}\ (P, \textit{hasta})) \end{array}$$

La notación $p : \textit{Avión}$ en la lista de parámetros es una abreviatura de $\textit{Avión}(p)$ en la sentencia de precondición; este cambio no añade valor expresivo, pero es más sencillo de leer (también reduce el número de posibles acciones proposicionales que pueden ser construidas). La precondición ($\textit{desde} \neq \textit{hasta}$) expresa el hecho de que un vuelo no discurre entre un aeropuerto y él mismo. Esto podría no haber sido expresado suficientemente por STRIPS.

Los diferentes formalismos de planificación usados en la IA han sido sistematizados dentro de una sintaxis estándar llamada Lenguaje de Definición de Dominios para la Planificación (*Planning Domain Definition Language*, o PDDL). Este lenguaje permite a los investigadores el intercambio y la comparación de problemas y resultados. PDDL incluye sublenguajes para STRIPS, ADL y para las redes jerárquicas de tareas que veremos en el Capítulo 12.

Lenguaje STRIPS	**Lenguaje ADL**
Sólo literales positivos en estados: $\textit{Pobre} \wedge \textit{Desconocido}$	Literales positivos y negativos en estados: $\neg\textit{Rico} \wedge \neg\textit{Famoso}$
Hipótesis de Mundo Cerrado: Los literales no mencionados son falsos	Hipótesis de Mundo Abierto: Los literales no mencionados son desconocidos
El efecto de $P \wedge \neg Q$ significa añadir P y eliminar Q	El efecto de $P \wedge \neg Q$ significa añadir P y $\neg Q$ y eliminar $\neg P$ y Q
Sólo literales simples en objetivos: $\textit{Rico} \wedge \textit{Famoso}$	Variables cuantificadas en objetivos: $\exists x\ En\ (P_1, x) \wedge En(P_2, x)$ es el objetivo de tener P_1 y P_2 en el mismo lugar
Los objetivos son conjunciones: $\textit{Rico} \wedge \textit{Famoso}$	Se permiten conjunciones y disyunciones en los objetivos: $\neg\textit{Pobre} \wedge (\textit{Famoso} \vee \textit{Inteligente})$
Los efectos son conjunciones	Se permiten efectos condicionales: **cuando** P: E significa que E es un efecto sólo si P es satisfecho
No tiene infraestructura para soportar igualdades	Predicados de igualdad ($x = y$) son admisibles
No tiene infraestructura para soportar tipos	Las variables pueden tener tipos, como (p: *Avión*)

Figura 11.1 Comparación de lenguajes STRIPS y ADL para la representación de problemas de planificación. En ambos casos, los objetivos se comportan como las precondiciones de una acción sin parámetros.

Tanto la notación STRIPS como ADL son adecuadas para muchos dominios reales. Las subsecciones que siguen muestran algunos ejemplos. Sin embargo, existen algunas restricciones significativas. La más obvia es que no pueden representar de un modo natural las **ramificaciones** de las acciones. Por ejemplo, si existe gente, paquetes o motas de polvo en el avión, todas éstas cambiarán de localización cuando el avión vuele. Podemos representar estos cambios como el efecto directo de volar, considerando que parece más natural representar la localización del contenido del avión como la consecuencia lógica de la localización del mismo. Veremos otros ejemplos de estados con **constricciones de estado** en la Sección 11.5. Los sistemas de planificación clásica no son capaces de tratar problemas de **requisitos:** el problema de no representar circunstancias que puedan causar que una acción fracase. Veremos cómo tratar con los requisitos en el Capítulo 12.

CONSTRICCIONES DE ESTADO

Ejemplo: transporte de carga aéreo

La Figura 11.2 muestra cómo un problema de transporte de carga aéreo lleva asociado procesos de carga y descarga entre aviones que vuelan entre diferentes destinos. El problema puede ser definido con tres acciones: *Carga, Descarga*, y *Vuelo*. Las acciones afectan a dos predicados: $Dentro(c, p)$ significa que la carga c está dentro del avión p, y $En(x, a)$ significa que el objeto x (tanto avión como carga) está en el aeropuerto a. Notemos que la carga no está *En* cualquier sitio cuando se encuentra *Dentro* de un avión concreto, por tanto *En* realmente significa «disponible para su uso en una localización determinada». El siguiente plan es una solución al problema:

$$[Carga(C_1, P_1, SFO), Volar(P_1, SFO, JFK),\\ Carga(C_2, P_2, JFK), Volar(P_2, JFK, SFO)]$$

Nuestra representación es estrictamente STRIPS. En particular, se permite que un avión vuele hacia y desde el mismo aeropuerto. Literales de tipo desigualdades, podrían prevenir este tipo de situaciones.

$Iniciar\ (En(C_1, SFO) \wedge En(C_2, JFK) \wedge En(P_1, SFO) \wedge En(P_2, JFK)$
$\quad \wedge Carga(C_1) \wedge Carga(C_2) \wedge Avión(P_1) \wedge Avión(P_2)$
$\quad \wedge Aeropuerto(JFK) \wedge Aeropuerto(SFO)$
$Objetivo(En(C_1, JFK) \wedge En(C_2, SFO))$
$Acción\ (Cargar(c, p, a),$
$\quad$ PRECOND: $En\ (c, a) \wedge En(p, a) \wedge Carga(c) \wedge Avión(p) \wedge Aeropuerto(a)$
$\quad$ EFECTO: $\neg En\ (c, a) \wedge Dentro\ (c, p)$
$Acción\ (Descargar(c, p, a),$
$\quad$ PRECOND: $Dentro\ (c, p) \wedge En\ (p, a) \wedge Carga(c) \wedge Avión(p) \wedge Aeropuerto(a)$
$\quad$ EFECTO: $En\ (c, a) \wedge \neg Dentro\ (c, p))$
$Acción\ (Volar(p, desde, hasta),$
$\quad$ PRECOND: $En\ (p, desde) \wedge Avión(p) \wedge Aeropuerto(desde) \wedge Aeropuerto(hasta)$
$\quad$ EFECTO: $\neg En\ (p, desde) \wedge En\ (p, hasta))$

Figura 11.2 Problema STRIPS de transporte de carga aérea entre aeropuertos.

Ejemplo: el problema de la rueda de recambio

Considérese el problema de cambiar una rueda pinchada. De modo más preciso, el objetivo es tener la rueda de repuesto montada correctamente en el eje del coche, mientras que el estado inicial consiste en la rueda pinchada sobre el eje y la rueda de repuesto en el maletero. Para exponerlo sencillamente, nuestra versión del problema es abstracta, sin considerar otras complicaciones. Simplemente contamos con cuatro acciones: sacar la rueda del maletero, quitar la rueda pinchada del eje, colocar la rueda nueva en el eje y dejar el coche sin vigilancia durante la noche. Se asume que el coche queda aparcado en un barrio peligroso, de modo que el efecto de dejarlo sin vigilancia puede ser el que roben las ruedas.

La descripción ADL del problema se muestra en la Figura 11.3. Notemos que es estrictamente proposicional. Vamos más allá que con el simple uso de lenguaje STRIPS, pues se usan precondiciones negativas, $\neg En(Deshinchada, Eje)$, para la acción $Colocar(Repuesto, eje)$. Esto podría ser evitado mediante la utilización de $Despejar(eje)$, como veremos en el siguiente ejemplo.

Iniciar (*En*(*Deshinchada, Eje*) ∧ *En*(*Repuesto, maletero*))
Objetivo(*En*(*Repuesto, eje*))
Acción (*Quitar,* (*Repuesto, maletero*),
  PRECOND: *En*(*Repuesto, maletero*)
  EFECTO: ¬*En*(*Repuesto, maletero*) ∧ *En*(*Repuesto, Suelo*))
Acción (*Quitar,* (*Deshinchada, Eje*),
  PRECOND: *En*(*Deshinchada, Eje*),
  EFECTO: ¬*En*(*Deshinchada, Eje*) ∧ *En*(*Deshinchada, Suelo*))
Acción (*Colocar*(*Repuesto, eje*).
  PRECOND: *En*(*Repuesto, Suelo*) ∧ ¬*En*(*Deshinchada, Eje*)
  EFECTO: ¬*En*(*Repuesto, Suelo*) ∧ *En*(*Repuesto, Eje*))
Acción (*DejarloDeNoche,*
  PRECOND:
  EFECTO: ¬*En*(*Repuesto, Suelo*) ∧ ¬*En*(*Repuesto, Eje*) ∧ ¬*En*(*Repuesto, maletero*)
    ∧ ¬*En*(*Deshinchada, Suelo*) ∧ ¬*En*(*Deshinchada, Eje*))

Figura 11.3 El problema simplificado de la rueda de repuesto.

Ejemplo: el mundo de los bloques

MUNDO DE LOS BLOQUES

Uno de los más famosos dominios de planificación es conocido como el **mundo de los bloques**. Este dominio consiste en un conjunto de bloques con forma de cubo situados en un tablero[3]. Los bloques pueden ser amontonados, pero únicamente podemos situar uno sobre otro. Un brazo mecánico puede cambiar bloques de sitio, tanto sobre el ta-

[3] El mundo de los bloques utilizado en planificación es mucho más simple que la versión SHRDLU 'S, mostrada en el Apartado 1.3.

blero como sobre otros bloques. El brazo puede tomar sólo un bloque por cada instante de tiempo, de modo que no puede tomar uno si aún no ha soltado otro anterior. El objetivo será construir uno o más montones de bloques, que quedarán especificados en términos de cuántos bloques están sobre cuántos otros bloques. Por ejemplo, podríamos tener un objetivo que fuese colocar un bloque *A* sobre otro *B* y uno *C* sobre otro *D*.

Usemos *Sobre*(*b, x*) para indicar que el bloque *b* se encuentra sobre *x,* donde *x* puede ser también otro bloque sobre el tablero. La acción para mover el bloque *b* desde el lugar *x* hasta el lugar *y* será *Mover*(*b, x, y*). Una de las precondiciones para poder mover *b* es que ningún otro bloque se encuentre sobre él. En una lógica de primer orden, esta idea podría ser expresada como $\neg\exists\, x\ Sobre(x, b)$ o, alternativamente, $\forall x\ \neg Sobre(x, b)$. Éstas podrían ser establecidas como precondiciones en ADL. Podemos expresarlas dentro de un lenguaje STRIPS, sin embargo, añadiendo un nuevo predicado, *Despejar*(*x*), que es verdadero siempre que ningún bloque esté sobre *x*.

La acción *Mover* cambia un bloque *b* desde *x* hasta *y* si tanto *b* como *y* están «despejados». Después de que el movimiento sea ejecutado, *x* estará despejado pero *y* no lo estará. Una descripción formal de *Mover* en STRIPS es:

$$
\begin{array}{l}
Acción(Mover\ (b, x, y),\\
\quad \text{PRECOND: } Sobre(b, x) \wedge Despejar(b) \wedge Despejar(y),\\
\quad \text{EFECTO: } Sobre(b, y) \wedge Despejar(x) \wedge \neg Sobre(b, x) \wedge \neg Despejar(y))
\end{array}
$$

Desafortunadamente, esta acción no mantiene *Despejar* adecuadamente cuando *x* o *y* se encuentran sobre el tablero. Cuando $x = Tablero$, esta acción tiene el efecto *Despejar*(*Tablero*), pero el tablero no debe estar vacío; de igual modo cuando $y = Tablero$. Para determinar estas situaciones, podemos hacer dos cosas. En primer lugar, incluimos otra acción para mover un bloque *b* desde *x* al tablero:

$$
\begin{array}{l}
Acción\ (\ MoverSobreTablero(b, x),\\
\quad \text{PRECOND: } Sobre(b, x) \wedge Despejar(b)),\\
\quad \text{EFECTO: } Sobre(b, Tablero) \wedge Despejar(x) \wedge \neg Sobre(b, x))
\end{array}
$$

$$
\begin{array}{l}
Iniciar(Sobre(A, Tablero) \wedge Sobre(B, Tablero) \wedge Sobre(C, Tablero)\\
\quad \wedge Bloque(A) \wedge Bloque\ (B) \wedge Bloque(C)\\
\quad \wedge Despejar(A) \wedge Despejar(B) \wedge Despejar(C))\\
Objetivo\ (Sobre(A, B) \wedge Sobre(B, C))\\
Acción\ (\ Mover\ (b, x, y),\\
\quad \text{PRECOND: } Sobre(b, x) \wedge Despejar(b) \wedge Despejar(y) \wedge Bloque\ (b) \wedge\\
\quad\quad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),\\
\quad \text{EFECTO: } Sobre(b, y) \wedge Despejar(x) \wedge \neg Sobre(b, x) \wedge \neg Despejar(y))\\
Acción\ (MoverSobreTablero(b, x),\\
\quad \text{PRECOND: } Sobre(b, x) \wedge Despejar(b) \wedge Bloque\ (b) \wedge (b \neq x),\\
\quad \text{EFECTO: } Sobre(b, Mesa) \wedge Despejar(x) \wedge \neg Sobre(b, x))
\end{array}
$$

Figura 11.4 Un problema de planificación en el mundo de los bloques: construir una torre de tres bloques. Una solución es la secuencia [*Mover*(*B, Tablero, C*), *Mover*(*A, Tablero, B*)].

En segundo lugar, tomaremos la interpretación de *Despejar*(*b*) como «existe un espacio vacío sobre *b* para trasladar un bloque». Bajo esta interpretación, *Despejar*(*Tablero*) siempre será correcto. El único problema es que nada le impide al planificador usar *Mover*(*b, x, Tablero*) en lugar de *MoverSobreTablero*(*b, x*). Podríamos vivir con este problema (nos llevará a un espacio de búsqueda mayor que el necesario, pero no se tendrán respuestas incorrectas) o podemos incluir el predicado *Bloque* y añadir *Bloque*(*b*) $\wedge$ *Bloque*(*y*) como precondición de *Mover*.

Finalmente, existe el problema de las acciones espurias tales como *Mover*(*B, C, C*), que tiene efectos contradictorios. Es común ignorar tales problemas, porque casi nunca provocan planes incorrectos. El enfoque correcto añade precondiciones de tipo desigualdad como se muestra en la Figura 11.4.

11.2 Planificación con búsquedas en espacios de estado

Prestemos atención a los algoritmos de planificación. El enfoque más sencillo es el uso de una búsqueda en un espacio de estados. Es conocido que las descripciones de las acciones en un problema de planificación especifican tanto las precondiciones como los efectos, por tanto son posibles las búsquedas en ambas direcciones: búsquedas hacia-adelante desde el estado inicial o búsquedas hacia-atrás desde el estado final, como se muestra en la Figura 11.5. También usaremos las representaciones de objetivos y acciones explícitas para derivar automáticamente heurísticas efectivas.

Búsquedas hacia-delante en el espacio de estados

La planificación mediante búsquedas hacia-delante en el espacio de estados es similar al enfoque que veíamos en el Capítulo 3 para resolver problemas. Algunas veces se le llama planificación de **progresión**, porque mantiene una dirección de avance.

PROGRESIÓN

Comenzamos en el estado inicial del problema, considerando secuencias de acciones hasta que encontremos una secuencia que alcance un estado objetivo. La formulación de problemas de planificación como problemas de búsqueda en un espacio de estados es como sigue:

- El **estado inicial** de la búsqueda es el estado inicial del problema de planificación. En general, cada estado será un conjunto de literales simples y positivos; los literales que no aparecen expresados se asume que son falsos.
- Las **acciones** que son aplicables en un estado son todas aquellas cuyas precondiciones son satisfechas. El estado resultante de una acción es generado añadiendo literales positivos y eliminando los negativos. Notemos que una simple función sucesor trabaja para todos los problemas de planificación (una consecuencia de utilizar una representación de acción explícita).
- El **test de objetivos** chequea si el estado satisface el objetivo del problema de planificación.

Figura 11.5 Dos enfoques para alcanzar un plan. (a) Búsqueda en el espacio de estados hacia-delante (progresión), comenzando en el estado inicial y usando las acciones del problema para avanzar hacia el estado objetivo. (b) Búsqueda en el espacio de estados hacia-atrás (regresión): una búsqueda en el espacio de creencias (*véase* Apartado 3.6) comenzando en los estados objetivo y utilizando el inverso de las acciones para rastrear hacia atrás hasta el estado inicial.

- El **coste del paso** entre acciones es típicamente 1. Aunque podría ser sencillo permitir diferentes costes para diferentes acciones, esto es rara vez utilizado por los planificadores STRIPS.

Recalquemos que, en ausencia de símbolos funcionales, el espacio de estados de un problema de planificación es finito. Por tanto, cualquier algoritmo de búsqueda en grafos que es completo (por ejemplo, A^*) será un algoritmo de planificación completo.

Desde los inicios de la investigación en planificación (en torno a 1961) hasta la actualidad (1998) se ha constatado que la búsqueda hacia-delante en un espacio de estados es ineficaz en la práctica. No es difícil argumentar esta posición con varios resultados, simplemente revisando la Sección 11.1. En primer lugar, las búsquedas hacia-delante no son capaces de tratar con el problema de acciones irrelevantes, todas las posibles acciones aplicables son consideradas desde el estado en el que nos encontramos. En segundo lugar, este enfoque rápidamente queda empantanado sin una buena heurística. Consideremos el problema del transporte de carga aéreo con los siguientes datos, 10 aeropuertos, cada aeropuerto posee cinco aviones y 20 piezas para ser cargadas. El objetivo es trasladar toda la carga desde el aeropuerto A al B. Existe una solución muy simple para este problema: descargar las 20 piezas en uno de los aviones en A, volar hasta B y proceder a su descarga. Sin embargo, encontrar la solución puede ser difícil porque el factor promedio de ramificación es enorme: cada uno de los 50 aviones puede volar a otros nueve aeropuertos, y cada uno de los 200 paquetes pueden ser tanto descargados (si antes

fueron cargados) o cargados dentro de cada avión en un aeropuerto (si se encuentran descargados). En promedio, tenemos unas 1.000 acciones posibles, de modo que el árbol de búsqueda para encontrar la solución obvia tiene del orden de 1.000^{41} nodos. Es evidente que una heurística adecuada al problema es necesaria para hacer este tipo de búsqueda eficiente. Discutiremos algunas posibles heurísticas después de presentar la búsqueda hacia-atrás.

Búsquedas hacia-atrás en el espacio de estados

Las búsquedas hacia-atrás en el espacio de estados fueron descritas brevemente como parte de búsquedas bidireccionales en el Capítulo 3. Destacamos, en primer lugar, que la búsqueda hacia-atrás puede ser difícil de implementar cuando el estado objetivo es descrito por un conjunto de constricciones más que siendo explícitamente especificado. En particular, no es siempre obvio cómo generar una descripción de los posibles estados **predecesores** del conjunto de estados objetivo. Veremos que la representación de un problema de planificación mediante STRIPS se simplifica porque el conjunto de estados puede ser descrito por el conjunto de literales que deben verificarse en ellos.

RELEVANCIA

La principal ventaja de las búsquedas hacia-atrás es que nos permiten considerar solamente acciones **relevantes**. Una acción es relevante para una secuencia encadenada de objetivos si alcanza un conjunto de ellos. Por ejemplo, el objetivo en nuestro problema de transporte de carga aérea considerando 10 aeropuertos, es tener 20 piezas de carga en el aeropuerto B, o de manera más precisa:

$$En(C_1, B) \wedge En(C_2, B) \wedge \ldots \wedge En(C_{20}, B)$$

Ahora consideremos el conjunto $En(C_1, B)$. Trabajando hacia-atrás, podremos obtener acciones que tengan ésta como efecto. Por ejemplo, una de ellas: $Descargar(C_1, p, B)$, donde el avión p no está especificada.

Notemos que existen muchas acciones *irrelevantes* que pueden dirigirnos hacia un estado objetivo. Por ejemplo, podemos volar en un avión vacío desde *JFK* hasta *SFO*; esta acción alcanza un estado objetivo desde un estado predecesor en el que el avión está en *JFK* y todo el conjunto de objetivos es satisfecho. Una búsqueda hacia-atrás que permita acciones irrelevantes será completa pero mucho menos eficiente. Si la solución existe, será encontrada por una búsqueda hacia atrás que permita solamente acciones relevantes. La restricción a acciones relevantes se traduce en búsquedas hacia atrás que tienen un factor *de ramificación* mucho menor que en las búsquedas hacia delante. Por ejemplo, nuestro problema de carga aérea tiene cerca de 1.000 acciones dirigidas hacia delante desde el estado inicial pero sólo 20 acciones hacia atrás desde el estado objetivo.

REGRESIÓN

Las búsquedas hacia atrás son conocidas habitualmente como planificación por **regresión**. La principal pregunta en la planificación por regresión es: ¿Cuáles son los estados desde los cuales iniciar una acción determinada que nos dirige hacia el objetivo? La descripción de estos estados se conoce como la regresión de un objetivo a través de la acción. Para ver cómo hacer esto, consideremos el ejemplo del transporte de carga aéreo. Tenemos el siguiente objetivo:

$$En(C_1, B) \wedge En(C_2, B) \wedge \ldots \wedge En(C_{20}, B)$$

y la acción relevante $Descargar(C_1, p, B)$, que alcanza el primer término. La acción será adecuada solamente si sus precondiciones son satisfechas. Sin embargo, cualquier estado predecesor debe incluir las precondiciones siguientes: $Dentro(C_1, p) \wedge En(p, B)$. Sin embargo, el subconjunto $En(C_1, B)$ no debería ser cierto en el estado predecesor[4]. De este modo, la descripción precedente es

$$Dentro(C_1, p) \wedge En(p, B) \wedge En(C_2, B) \wedge \ldots \wedge En(C_{20}, B)$$

Además de insistir en que las acciones alcancen algunos literales deseados, debemos insistir en que las acciones *no deshagan* ningún literal deseado. Una acción que satisfaga dicha restricción es llamada **consistente**. Por ejemplo, la acción $Cargar(C_2, p)$ podría no ser consistente con el objetivo actual, porque podría negar el literal $En(C_2, B)$.

CONSISTENCIA

Dadas definiciones de relevancia y consistencia, podemos describir el proceso general de construcción de predecesores mediante búsquedas hacia-atrás. Sea la descripción de un objetivo G, y supongamos que A sea una acción que es relevante y consistente. El correspondiente predecesor es como sigue:

- Cualquier efecto positivo de A que aparezca en G es eliminado.
- Cada precondición literal de A es añadida, a no ser que ya apareciese.

Cualquiera de los algoritmos de búsqueda estándar pueden ser usados para llevar adelante la búsqueda. Se finaliza el proceso cuando la descripción de un predecesor generada es satisfecha por el estado inicial del problema de planificación. En el caso de primer-orden, la satisfacción puede requerir la sustitución por variables en la descripción del precedente. Por ejemplo, la descripción del precedente en el párrafo anterior es satisfecha por el estado inicial

$$Dentro(C_1, P_{12}) \wedge En(P_{12}, B) \wedge En(C_2, B) \wedge \ldots \wedge En(C_{20}, B)$$

sustituyendo $\{p/P_{12}\}$. La sustitución debe ser aplicada a las acciones que se dirigen hacia el objetivo, produciendo la solución $[Descarga(C_1, P_{12}, B)]$.

Heurísticas para la búsqueda en el espacio de estados

Se ha indicado previamente que tanto para que algoritmos de búsqueda hacia-delante como hacia-atrás sean eficientes deben utilizar una función heurística adecuada. Recordemos del Capítulo 4 que una función heurística estima la distancia desde un estado al objetivo; en la planificación STRIPS, el coste de cada acción es 1, de modo que la distancia es el número de acciones. La idea básica es observar los efectos de las acciones y los objetivos que deben ser alcanzados y estimar cuántas acciones son necesarias para alcanzar todos los objetivos. Encontrar el número exacto es un problema NP-completo, pero es posible encontrar razonables estimaciones en muchos casos sin demasiado gasto computacional. Podríamos, de igual modo, ser capaces de derivar una heurística **admisible**, esto es, que no sobrestime. Podríamos usar A* para encontrar soluciones óptimas.

[4] Si el subobjetivo fuese verdadero en el estado predecesor, la acción podría aún dirigirse hasta el estado objetivo. Por otro lado, tales acciones son irrelevantes porque no *hacen* el objetivo verdadero.

Existen dos enfoques que deben ser mencionados. El primero consiste en derivar un **problema aproximado** desde las especificaciones del problema dado, tal como se describió en el Capítulo 4. El coste de la solución óptima del problema aproximado (que suponemos sencillo de resolver) nos da una heurística admisible para el problema original. El segundo enfoque pretende usar un simple algoritmo divide-y-vencerás. Este planteamiento asume la **independencia de sub-objetivos**: el coste de resolver una secuencia de sub-objetivos es aproximadamente la suma de los costes que supone resolver cada uno de los sub-objetivos *independientemente*. La hipótesis de la independencia de sub-objetivos puede ser optimista o pesimista. Se llama optimista cuando existen interacciones negativas entre los subplanes de cada sub-objetivo, por ejemplo, cuando una acción en un sub-plan hace fracasar el objetivo a alcanzar por otro sub-plan. Es pesimista, y por tanto inadmisible, cuando los sub-planes contienen acciones redundantes, por ejemplo, dos acciones que podrían ser reemplazadas por una acción sencilla en un plan conjunto.

INDEPENDENCIA DE SUB-OBJETIVOS

Consideremos cómo obtener problemas aproximados a partir de uno dado. Si tenemos disponibles las representaciones explícitas de las precondiciones y de los efectos, el proceso consistirá en la modificación de dichas representaciones (comparar este enfoque con los problemas de búsqueda, donde la función sucesora es una caja negra). La idea más simple para calcular un problema aproximado es mediante la *eliminación de todas las precondiciones* de las acciones. En esta situación, todas las acciones serán aplicables, y cualquier literal puede ser alcanzado en cada etapa (si existe una acción que es aplicable, en caso contrario, el objetivo es imposible). Esto prácticamente implica que el número de pasos requeridos para resolver una secuencia de objetivos es el número de objetivos no satisfechos, prácticamente pero no totalmente porque (1) puede haber dos acciones, cada una de las cuales elimine el literal del objetivo alcanzado por el otro, y (2) alguna acción puede alcanzar múltiples objetivos. Si combinamos el problema aproximado junto con la hipótesis de la independencia de sub-objetivos, ambas se encuentran lejos de que la heurística resultante sea exactamente el número de objetivos no satisfechos por alcanzar.

En algunos casos, una heurística más adecuada se obtiene mediante la consideración de interacciones positivas que surgen de las acciones que permiten alcanzar múltiples objetivos. En primer lugar, derivamos el problema aproximado mediante la *eliminación de efectos negativos* (*véase* Ejercicio 11.6). Después, contamos el mínimo número de acciones requeridas tales que la unión de los efectos de acciones positivas satisfagan el objetivo. Por ejemplo, consideremos

$$\begin{aligned}
&\textit{Objetivo}(A \wedge B \wedge C)\\
&\textit{Acción}(X, \text{EFECTO}: A \wedge P)\\
&\textit{Acción}(Y, \text{EFECTO}: B \wedge C \wedge Q)\\
&\textit{Acción}(Z, \text{EFECTO}: B \wedge P \wedge Q)
\end{aligned}$$

El mínimo conjunto que cubre el objetivo $\{A, B, C\}$ viene dado por las acciones $\{X, Y\}$, de modo que el conjunto que cubre la heurística devuelve un coste de 2. Esto mejora la hipótesis de la independencia de sub-objetivos, cuya heurística nos da un valor de 3. Existe un pequeño inconveniente: el problema es NP-duro. Un algoritmo simple que cubra el conjunto garantiza el retorno de un valor que esté dentro de un orden *log n* del valor mínimo verdadero, donde n es el número de literales en el objetivo; normalmente fun-

ciona bien en la práctica. Desafortunadamente un algoritmo de este tipo pierde la garantía de admisibilidad para la heurística.

Es también posible generar problemas aproximados eliminando efectos negativos sin eliminar precondiciones. Esto es, si una acción tiene el efecto $A \wedge \neg B$ en el problema original, tendrá el efecto A en el problema aproximado. Esto significa que no necesitamos preocuparnos acerca de la interacciones negativas entre sub-planes, porque ninguna acción puede eliminar los literales alcanzados por otra acción. El coste de la solución del resultante problema aproximado da lo que se conoce como heurística para **suprimir listas vacías**. La heurística es bastante precisa, pero ponerla a funcionar lleva asociado ejecutar un algoritmo de planificación simple. En la práctica, la búsqueda en el problema aproximado es suficientemente rápida como para que el coste merezca la pena.

SUPRIMIR LISTAS VACIAS

Las heurísticas descritas aquí pueden ser usadas tanto en dirección de progresión como regresión. Actualmente, los planificadores de progresión que usan la heurística para suprimir listas vacías marcan la tendencia. Es probable que se produzcan cambios y que nuevas heurísticas y nuevas técnicas de búsqueda sean exploradas. Dado que la planificación es exponencialmente difícil[5], ningún algoritmo será eficiente para todos los problemas, pero una gran cantidad de problemas prácticos pueden ser resueltos con los métodos heurísticos mencionados en este capítulo, muchos más que los que se podían resolver solamente hace unos pocos años.

11.3 Planificación ordenada parcialmente

Las búsquedas en el espacio de estados hacia-delante y hacia-atrás son tipos de planes de búsqueda *totalmente ordenados*. Sólo exploran secuencias estrictamente lineales de acciones conectadas directamente al inicio o al objetivo. Esto significa que no se puede sacar provecho de la descomposición del problema. Preferiblemente que trabajar sobre cada subproblema separadamente, se deben tomar decisiones sobre el orden en que se sucedan las acciones desde todos los subproblemas. Preferiremos un enfoque que trabaje en varios sub-objetivos independientemente, los solucione con varios sub-planes, y por último, combine el conjunto de sub-planes utilizados.

Este enfoque presenta la ventaja de flexibilizar el orden en el que se *construye* el plan. Es decir, el planificador puede trabajar sobre «obvias» o «importantes» decisiones primeramente, antes que estar forzado a trabajar siguiendo las etapas en un orden cronológico. Por ejemplo, un agente planificador que se encuentre en Berkeley y desee viajar a Monte Carlo podría primero intentar encontrar un vuelo desde San Francisco a París; dada información acerca de los horarios de partida y llegada, puede ponerse a trabajar en los modos de salir y llegar a los aeropuertos.

MÍNIMO COMPROMISO

La estrategia general de aplazar una opción durante la búsqueda se conoce como una estrategia de **mínimo compromiso**. No damos una definición formal, aunque claramente

[5] Técnicamente, la planificación tipo STRIPS es PSPACE-completa a menos que tengan sólo precondiciones positivas y sólo un literal como efecto (Bylander, 1994).

un grado de compromiso es necesario, dado que si no, la búsqueda podría no progresar. Dejando a un lado esta informalidad, el mínimo compromiso es un concepto útil para analizar cuándo las decisiones deben llevarse a cabo en cualquier problema de búsqueda.

Nuestro primer ejemplo concreto será más simple que planificar unas vacaciones. Consideremos el sencillo problema de ponerse un par de zapatos. Podemos describirlo como un problema de planificación formal como sigue:

Objetivo (*ZapatoDerechoPuesto* ∧ *ZapatoIzquierdoPuesto*)
Inicio( )
Acción(*ZapatoDerecho,*
  PRECOND: *CalcetínDerechoPuesto,*
  EFECTO: *ZapatoDerechoPuesto*)
Acción(*CalcetínDerecho,*
  EFECTO: *CalcetínDerechoPuesto*)
Acción(*ZapatoIzquierdo,*
  PRECOND: *CalcetínIzquierdoPuesto,*
  EFECTO: *ZapatoIzquierdoPuesto*)
Acción(CalcetínIzquierdo,
  EFECTO: *CalcetínIzquierdoPuesto)*

Un planificador debe ser capaz de trabajar con las secuencias de dos-acciones *CalcetínDerecho* seguido por *ZapatoDerecho* para alcanzar el primer conjunto de objetivos, y la secuencia *CalcetínIzquierdo* seguido de *ZapatoIzquierdo* para el segundo conjunto. Por tanto, las dos secuencias pueden ser combinadas para cumplir el plan total. En este desarrollo, el planificador manipulará las dos secuencias independientemente, sin preocuparse de si una acción pertenece a una secuencia o a otra. Cualquier algoritmo de planificación que pueda conjuntar dos acciones dentro del mismo plan sin necesidad de conocer cuál de ellas es previa a la otra, se conoce como **planificador de primer orden.** La Figura 11.6. nos muestra un plan de orden parcial que constituye la solución para el problema de los zapatos y los calcetines. Destaquemos que la solución es representada como un *grafo* de acciones y no como una secuencia. Destaquemos también que las acciones «dummy», las acciones *Inicio* y *Final*, marcan el principio y el final del plan. Llamarlas acciones simplifica el tratamiento del problema, porque de este modo cada una de las etapas del plan es una acción. La solución de orden-parcial se corresponde con seis posibles planes de orden total; cada uno de ellos se conoce como una **linealización** del plan de orden parcial.

PLANIFICADOR DE PRIMER ORDEN

LINEALIZACIÓN

El planificador de orden-parcial puede ser implementado como una búsqueda en el espacio de los planes de orden parcial (desde ahora, los llamaremos únicamente «planes»). Esto es, comenzaremos con un plan vacío. Posteriormente consideraremos formas de refinar este plan hasta que tengamos un plan completo que resuelve el problema. Las acciones en esta búsqueda no son acciones en el mundo, sino acciones sobre planes: añadir una etapa a un plan, imponer una ordenación que coloca una acción antes de otra, etc.

Definiremos el algoritmo POP para procesos de planificación de orden-parcial. Es tradicional escribir el algoritmo POP como un programa autónomo, preferible a la formulación de planes de orden parcial como ejemplos de problemas de búsqueda. Esto nos permitirá centrarnos sobre los refinamientos de las etapas del plan que pueden ser apli-

Figura 11.6 Un plan de orden parcial para ponerse calcetines y zapatos, y las seis linealizaciones correspondientes dentro de los planes de orden total.

cados, en lugar de preocuparnos de cómo los algoritmos exploran el espacio. De hecho, una amplia variedad de métodos de búsqueda heurísticos poco fundamentados pueden ser aplicados una vez que el problema de búsqueda es formulado.

Recordemos que los estados de nuestros problemas de búsqueda serán mayoritariamente planes inacabados. Para evitar confusiones con los estados del mundo, nos referiremos a planes más que a estados. Cada plan tiene los siguientes cuatro componentes, donde los dos primeros definen las etapas del plan y los dos últimos sirven como función de contabilidad para determinar cómo los planes pueden ser extendidos:

- Un conjunto de **acciones** que confeccionen las etapas del plan. Éstas son tomadas del conjunto de acciones en el problema de planificación. El plan «vacío» contiene simplemente las acciones *Iniciar* y *Finalizar*. *Iniciar* no posee precondiciones y tiene como efectos los literales en el estado inicial del problema de planificación. *Finalizar* no tiene efectos y tiene como precondiciones los literales del objetivo del problema de planificación.

RESTRICCIONES ORDENADAS

- Un conjunto de **restricciones ordenadas**. Cada limitación ordenada es de la forma $A \prec B$, la cual se lee como «A antes de B» y significa que la acción A debe ser ejecutada en algún momento antes de B, pero no necesariamente en el estado inmediatamente anterior. Las restricciones ordenadas deben describir un orden parcial apropiado. Cualquier ciclo (del tipo $A \prec B$ y $B \prec A$) representa una contradicción, de modo que una limitación de orden no podrá ser añadida a un plan si crea un ciclo.

RELACIONES CAUSALES

ALCANCES

- Un conjunto de **relaciones causales**. Un enlace causal entre dos acciones A y B en un plan es escrito como $A \xrightarrow{p} B$ y se lee como «A **alcanza** B a través de p». Por ejemplo, el enlace causal

$$CalcetínDerecho \xrightarrow{CalcetínDerechoPuesto} ZapatoDerecho$$

expresa que *CalcetínDerechoPuesto* es un efecto de la acción *CalcetínDerecho* y una precondición para *ZapatoDerecho*. También nos informa de que *CalcetínDerechoPuesto* debe ser cierto durante el tiempo de acción que discurre desde la acción *CalcetínDerecho* a la acción *ZapatoDerecho*. En otras palabras, el plan no podría ser extendido mediante la aportación de una nueva acción C que crease un **conflicto** con el enlace causal. Una acción C entraría en conflicto con $A \xrightarrow{p} B$ si C tiene el efecto $\neg p$, y si C pudiera (de acuerdo con el conjunto de restricciones ordenadas) traer A antes que B. Algunos autores lo llaman **intervalos de protección** de enlaces causales, porque el link $A \xrightarrow{p} B$ protege a p de ser negado a lo largo del intervalo que va desde A hasta B.

CONFLICTOS

PRECONDICIONES ABIERTAS

- Un conjunto de **precondiciones abiertas**. Una precondición es abierta si no es alcanzada por ninguna acción en un plan. Los planificadores trabajan para reducir el conjunto de precondiciones abiertas al conjunto vacío, sin introducir contradicciones.

Por ejemplo, el plan definitivo de la Figura 11.6 tiene los siguientes componentes:

Acciones: {*CalcetínDerecho, ZapatoDerecho, CalcetínIzquierdo, ZapatoIzquierdo, Iniciar, Finalizar*}

Relaciones de orden: {$CalcetínDerecho \prec ZapatoDerecho$, $CalcetínIzquierdo \prec ZapatoIzquierdo$}

Enlaces: {$CalcetínDerecho \xrightarrow{CalcetínDerechoPuesto} ZapatoDerecho$,
$CalcetínIzquierdo \xrightarrow{CalcetínIzquierdoPuesto} ZapatoIzquierdo$,
$ZapatoDerecho \xrightarrow{ZapatoDerechoPuesto} Finalizar$,
$ZapatoIzquierdo \xrightarrow{ZapatoIzquierdoPuesto} Finalizar$}

PrecondicionesAbiertas: { }

PLAN CONSISTENTE

Definimos un **plan consistente** como un plan en el cual no hay ciclos en las restricciones ordenadas y no existen conflictos con los enlaces causales. Un plan consistente con precondiciones no abiertas es una **solución.** El lector debe estar convencido del siguiente hecho*: cada linealización de una solución de orden-parcial es una solución de orden-total cuya ejecución desde el estado inicial alcanza el estado objetivo*. Esto significa que podremos extender la noción de «ejecución de un plan» de planes de orden total a planes de orden parcial. Un plan de orden parcial es ejecutado por la elección de alguna de las acciones posibles. Veremos en el Capítulo 12 que la flexibilidad disponible para un agente cuando ejecuta un plan puede ser útil si el entorno no coopera. La flexibilidad en la ordenación también hace más sencillo combinar pequeños planes dentro de otros mayores, porque cada uno de los pequeños puede reordenar sus acciones para evitar conflicto con otros planes.

Ahora estamos preparados para formular el problema de búsqueda que POP resuelve. Comenzaremos con una formulación adecuada para problemas de planificación proposicional, dejando para más tarde las complicaciones derivadas de formulaciones de primer orden. Como es habitual, la definición incluye el estado inicial, las acciones y la evaluación del objetivo.

- El plan inicial contiene *Iniciar* y *Finalizar*, la restricción de orden *Iniciar* $\prec$ *Finalizar*, sin enlaces causales y todas las precondiciones en *Finalizar* como precondiciones abiertas.
- La función sucesora de manera arbitraria selecciona una precondición abierta p sobre una acción B y genera un plan sucesor para cada posible modo de selección consistente de una acción A que alcance p. La consistencia es impuesta como sigue:
  1. El enlace causal $A \xrightarrow{p} B$ y la restricción de orden $A \prec B$ son añadidos al plan. A puede ser una acción que ya existe en el plan o una nueva. Si es nueva, se añade al plan junto a las condiciones *Iniciar* $\prec A$ y $A \prec$ *Finalizar.*
  2. Resolvemos conflictos entre el nuevo enlace causal y el resto de acciones existentes y entre la acción A (si es nueva) y el resto de enlaces causales existentes. Un conflicto entre $A \xrightarrow{p} B$ y C es resuelto haciendo que C ocurra en algún momento fuera de la protección del intervalo, tanto añadiendo $B \prec C$ o $C \prec A$.
- La evaluación del objetivo chequea si un plan es una solución para el problema de planificación original. Como solamente son generados planes consistentes, la evaluación del objetivo simplemente necesita que no existan precondiciones abiertas.

Recordemos que las acciones consideradas por los algoritmos de búsqueda bajo esta formulación son etapas de refinamiento de planes más que acciones reales del propio dominio. El coste del camino es irrelevante, estrictamente hablando, porque lo único que nos preocupa es el coste total de las acciones reales en el plan llevado a cabo. Sin embargo, *es* posible especificar una función de coste del camino que refleje los costes reales del plan: computamos 1 por cada acción real añadida al plan y 0 para todo el resto de etapas de refinamiento. De este modo, $g(n)$, donde n representa un plan, será igual al número de acciones reales en el plan. Una estimación heurística $h(n)$ puede también ser usada.

A primera vista, uno podría pensar que la función sucesora debería incluir sucesores *para cada p* abierta, y no simplemente para uno de ellos. Esto podría ser redundante e ineficaz, sin embargo, por la misma razón, los algoritmos de satisfacción de restricciones no incluyen sucesores para cada variable posible: el orden en el que consideremos las precondiciones abiertas (como el orden en el que consideramos variables CSP) es conmutativo (*véase* Apartado 5.2). De este modo, podemos elegir una ordenación arbitraria y aún tener un algoritmo completo. La elección del orden correcto puede llevarnos a una búsqueda más rápida, pero todas las ordenaciones finalizan con el mismo conjunto de soluciones candidatas.

Ejemplo de planificación de orden parcial

Veamos cómo POP resuelve el problema de la rueda de repuesto de la Sección 11.2. La descripción del problema es repetida en la Figura 11.7.

Iniciar (*En*(*Deshinchada, Eje*) ∧ *En*(*Repuesto, Maletero*))
Objetivo(*En*(*Repuesto, Eje*))
Acción (*Quitar*(*Repuesto, Maletero*),
  PRECOND: *En*(*Repuesto, Maletero*)
  EFECTO: ¬*En*(*Repuesto, Maletero*) ∧ *En*(*Repuesto, Suelo*))
Acción (*Quitar*(*Deshinchada, Eje*),
  PRECOND: *En*(*Deshinchada, Eje*),
  EFECTO: ¬*En*(*Deshinchada, Eje*)∧ *En*(*Deshinchada, Suelo*))
Acción (*Colocar*(*Repuesto, Eje*),
  PRECOND: *En*(*Repuesto, Suelo*) ∧ ¬*En*(*Deshinchada, Eje*),
  EFECTO: ¬*En*(*Repuesto, Suelo*) ∧ *En*(*Repuesto, Eje*))
Acción (*DejarloDeNoche*,
  PRECOND:
  EFECTO: ¬*En*(*Repuesto, Suelo*) ∧ ¬*En*(*Repuesto, Eje*) ∧ ¬*En*(*Repuesto, Maletero*)
    ∧ ¬*En*(*Deshinchada, Suelo*) ∧ ¬*En*(*Deshinchada, Eje*))

Figura 11.7 Descripción del problema simplificado de la rueda de repuesto.

La búsqueda de una solución comienza con el plan inicial, que contiene una acción *Iniciar* con el efecto *En*(*Repuesto, Maletero*) ∧ *En*(*Deshinchada, Eje*) y una acción *Finalizar* con la única precondición *En*(*Repuesto, Eje*). Entonces, generamos sucesores mediante la elección de una precondición abierta sobre la que trabajar (irrevocablemente) y la elección de diferentes acciones posibles para alcanzarlo. Hasta ahora, no nos preocupamos acerca de una función heurística que nos ayude en la toma de estas decisiones; aparentemente tendremos elecciones arbitrarias. La secuencia de eventos es como sigue:

1. Seleccionar la única precondición abierta de *Finalizar*, esto es, *En*(*Repuesto, Maletero*). Elegir la única acción aplicable, *Colocar*(*Repuesto, Eje*).
2. Seleccionar *En*(*Repuesto, Suelo*) precondición de *Colocar*(*Repuesto, Eje*). Elegir la única acción aplicable, *Quitar*(*Repuesto, Maletero*) para lograrlo. El plan resultante se muestra en la Figura 11.8.

Figura 11.8 Un plan de orden parcial incompleto para el problema de repuesto, después de seleccionar acciones para las primeras dos precondiciones abiertas. Las cajas representan acciones, las precondiciones a la izquierda y los efectos a la derecha. (Los efectos son omitidos, excepto para el de la acción *Iniciar*.) Las flechas oscuras representan enlaces causales protegiendo la proposición que indica la flecha.

3. Seleccionar ¬*En*(*Deshinchada, Eje*), precondición de *Colocar*(*Repuesto, Eje*). En sentido opuesto, seleccionar la acción *DejarloSinProtecciónNoche* en lugar de la acción *Quitar*(*Repuesto, Eje*). Destacamos que *DejarloSinProtecciónNoche* también tiene el efecto ¬*En*(*Repuesto, Suelo*), lo cual significa que entra en conflicto con el enlace causal

$$Quitar(Repuesto, Maletero) \xrightarrow{En(Repuesto,\ Suelo)} Colocar(Repuesto, Eje)$$

Para resolver el conflicto añadimos una limitación de orden colocando *DejarloSinProtecciónNoche* antes de *Quitar*(*Repuesto, Maletero*). El plan resultante es mostrado en la Figura 11.9 (¿Por qué esto resuelve el conflicto y por qué no existe otro modo de resolverlo?)

Figura 11.9 El plan después de seleccionar *DejarloSinProtecciónNoche* con la acción ¬*En*(*Deshinchada, Eje*). Para evitar un conflicto con el enlace causal *Quitar*(*Repuesto, Eje*) que protege *En*(*Repuesto, Suelo*), *DejarloSinProtecciónNoche* queda restringido a ocurrir antes de *Quitar*(*Repuesto, Maletero*), como es indicado por la flecha discontinua.

4. La única precondición abierta que permanece en este momento es *En*(*Repuesto, Maletero*). La única acción que puede alcanzarla es la acción *Iniciar*, pero el enlace causal entre *Iniciar* y *Quitar*(*Repuesto, Maletero*) está en conflicto con ¬*En*(*Repuesto, Maletero*), efecto de *DejarloSinProtecciónNoche:* no se puede ordenar antes que *Quitar*(*Repuesto, Maletero*), porque ya existe una restricción de orden antes de *Quitar*(*Repuesto, Maletero*). De este modo, estamos forzados a volver hacia atrás, eliminar la acción *Quitar*(*Repuesto, Maletero*) y los dos últimos enlaces causales, regresando al estado de la Figura 11.8. En esencia, el planificador ha demostrado que *DejarloSinProtecciónNoche,* no funciona como una forma de cambiar una rueda.
5. Consideremos de nuevo ¬*En*(*Deshinchada, Eje*), precondición de *Colocar*(*Repuesto, Eje*). En este momento, seleccionamos *Quitar*(*Deshinchada, Eje*).
6. Una vez más, seleccionamos *En*(*Repuesto, Rueda*) precondición de *Quitar*(*Repuesto, Maletero*) y seleccionar *Iniciar* para alcanzarlo. Esta vez no existen conflictos.
7. Seleccionar *En*(*Deshinchada, Eje*) precondición de *Quitar*(*Deshinchada, Eje*), y elegir *Iniciar* para alcanzarlo. Esto nos da un plan completo, consistente (en otras palabras, una solución) como mostramos en la Figura 11.10.

Figura 11.10 La solución final al problema de la rueda de repuesto. Notemos que *Quitar*(*Repuesto, Maletero*) y *Quitar*(*Deshinchada, Eje*), puede ser hecho en cualquier orden si son completados antes con la acción *Colocar*(*Repuesto, Eje*).

Aunque este ejemplo es muy simple, ilustra algunos de los puntos fuertes de la planificación de orden-parcial. En primer lugar, los enlaces causales nos llevan pronto a la poda de porciones del espacio de búsqueda que, por causa de conflictos irresolubles, no contiene soluciones. En segundo lugar, la solución en la Figura 11.10 es un plan de orden-parcial. En este caso la ventaja es pequeña, porque solamente existen dos posibles linealizaciones; sin embargo, un agente podría agradecer la flexibilidad, por ejemplo, si la rueda tiene que ser cambiada en la mitad de una carretera con tráfico denso.

El ejemplo también apunta a algunas posibles mejoras que podrían ser hechas. Por ejemplo, existe duplicación de esfuerzo: *Iniciar* se enlaza con *Quitar*(*Repuesto, Maletero*) antes que el conflicto cause un retroceso y sea entonces desconectado incluso aunque no esté implicado en el conflicto. Es entonces conectado de nuevo y la búsqueda continúa. Esta situación es típica de retrocesos cronológicos y podría estar atenuada por retrocesos dirigidos a dominios.

Planificación de orden parcial con variables independientes

En esta sección, consideraremos las complicaciones que pueden surgir cuando POP es usado con representación de acciones de primer orden que incluyen variables. Supongamos que tenemos un problema en un mundo de bloques (Figura 11.4) con las precondiciones abiertas $Sobre(A, B)$ y la acción:

$$\begin{aligned}&Acción\ (\ Mover\ (b, x, y),\\ &\quad \text{PRECOND}: Sobre(b, x) \wedge Despejar(b) \wedge Despejar(y),\\ &\quad \text{EFECTO}: Sobre(b, y) \wedge Despejar(x) \wedge \neg Sobre(b, x) \wedge \neg Despejar(y))\end{aligned}$$

Esta acción alcanza $Sobre(A, B)$ porque el efecto $Sobre(b, y)$ unifica $Sobre(A, B)$ con la sustitución $\{b/A, y/B\}$. Aplicando esta sustitución en la acción, nos queda:

$$\begin{aligned}&Acción\ (\ Mover\ (A, x, B),\\ &\quad \text{PRECOND}: Sobre(A, x) \wedge Despejar(A) \wedge Despejar(B),\\ &\quad \text{EFECTO}: Sobre(A, B) \wedge Despejar(x) \wedge \neg Sobre(A, x) \wedge \neg Despejar(B))\end{aligned}$$

Esto nos deja la variable x sin explicitar. Es decir, la acción nos habla de mover un bloque A desde algún lugar, sin decir cuál. Este es otro ejemplo del último principio de com-

promiso: podemos retrasar la toma de decisión hasta que alguna otra etapa en el plan lo haga por nosotros. Por ejemplo, supongamos que tenemos *Sobre*(*A, D*) en le estado inicial. Entonces la acción *Iniciar* puede ser usada para alcanzar *Sobre*(*A, x*), vinculando *x* a *D*. Esta estrategia que espera más información antes de fijar *x* es normalmente más eficiente que intentarlo para cualquier valor posible de *x* y hacer regresiones para cada una de las que fracase.

La presencia de variables en las precondiciones y en las acciones complica los procesos en la detección y resolución de conflictos. Por ejemplo, cuando *Mover* (*A, x, B*) es añadido al plan, necesitaremos una relación causal

$$Mover\ (A, x, B\ ) \xrightarrow{Sobre(A,B)} Finalizar$$

Si existe otra acción M_2 con efecto $\neg Sobre(A, z)$, entonces M_2 entra en conflicto sólo si *z* es *B*. Para acomodar esta posibilidad, extendemos la representación de planes con el fin de incluir un conjunto de **restricciones de desigualdad** de la forma $z \neq X$ donde *z* es una variable y *X* otra variable o una constante. En este caso, podríamos resolver el conflicto añadiendo $z \neq B$, lo que significa que las extensiones futuras al plan pueden instanciar *z* a cualquier valor excepto a *B*. En cualquier momento que apliquemos una sustitución al plan, debemos asegurarnos que las desigualdades no contradicen la sustitución. Por ejemplo, una sustitución que incluya x/y entra en conflicto con la inecuación $x \neq y$. Este tipo de conflictos no pueden ser resueltos, por tanto el planificador debe ejecutar regresiones en su desarrollo.

RESTRICCIONES DE DESIGUALDAD

Un ejemplo más exhaustivo de la planificación POP con variables en el mundo de los bloques viene dado en la Sección 12.6.

Heurísticas para planificación de orden parcial

Comparado con la planificación de orden total, la planificación de orden parcial posee una clara ventaja por su capacidad de descomponer problemas en subproblemas. Tiene también la desventaja de que no representa los estados directamente, de modo que es más difícil estimar cuánto de alejado está un plan de orden parcial de alcanzar un objetivo.

La heurística más obvia es el recuento del número de precondiciones abiertas distintas. Esto puede ser mejorado mediante la eliminación del número de precondiciones abiertas que se ajustan a los literales en el estado *Iniciar*. En un caso de orden total, esto sobrestima el coste cuando existen acciones que alcanzan múltiples objetivos y subestima el coste si existen interacciones negativas entre etapas del plan. La siguiente sección presenta un enfoque que nos permite obtener heurísticas más adecuadas para un problema aproximado.

La función heurística es usada para seleccionar qué plan refinar. Dada esta elección, el algoritmo genera sucesores basados en la selección de una única precondición abierta para trabajar sobre ella. En el caso de la selección variable en algoritmos de satisfacción de restricciones, esta selección tiene un impacto importante sobre la eficiencia. La **heurística más restrictiva** de CSPs puede ser adaptada para algoritmos de planificación y funcionar adecuadamente. La idea es seleccionar la condición abierta que pueda ser satisfecha por el *menor número* de caminos. Existen dos casos especiales para

esta heurística: primero, si una condición abierta no puede ser alcanzada por ninguna acción, la heurística la seleccionará. Esta es una buena estrategia porque la temprana detección de imposibilidad puede ahorrar un gran trabajo perdido. Segundo, si una condición abierta puede ser alcanzada de un único modo, debe ser seleccionada porque la decisión es inevitable y podría proporcionar restricciones adicionales sobre otras opciones aún sin ser hechas. Aunque la computación completa del número de formas de satisfacer cada condición abierta es costosa y no siempre merece la pena, los experimentos demuestran que trabajar con estos dos casos especiales proporciona una agilización sustanciosa.

11.4 Grafos de planificación

GRAFO DE PLANIFICACIÓN

Todas las heurísticas que hemos sugerido para planificación de orden parcial y total pueden sufrir imprecisiones. Esta sección muestra cómo una estructura especial llamada **grafo de planificación** puede ser usada para dar mejores estimaciones heurísticas. Estas heurísticas pueden ser aplicadas a cualquiera de las técnicas de búsqueda que hemos visto hasta ahora. Alternativamente, podemos extraer una solución directamente del grafo de planificación, usando un algoritmo especializado llamado GRAPHPLAN.

NIVELES

Un grafo de planificación consiste en una secuencia de **niveles** que corresponden a escalones de tiempo en el plan, y donde el nivel 0 es el estado inicial. Cada nivel contiene un conjunto de literales y un conjunto de acciones. A grandes rasgos, los literales son todos aquellos que *pueden* ser ciertos en cualquiera de las etapas, dependiendo solamente de las acciones ejecutadas en las etapas previas. También a grandes rasgos, las acciones son todas aquellas que *pueden* tener todas sus precondiciones satisfechas en cualquiera de las etapas, dependiendo de cuáles sean los literales sobre los que realmente actúan. Decimos a «grandes rasgos», porque el grafo de planificación solamente registra un restringido subcobjunto de posibles interacciones negativas entre sus acciones; por tanto, se puede ser optimista acerca del mínimo número de etapas temporales que se requieren para que un literal sea correcto. Sin embargo, el número de etapas en el grafo de planificación proporciona una buena estimación sobre la dificultad de alcanzar desde el estado inicial un literal dado. De manera más importante, el grafo de planificación es definido de tal manera que puede ser construido muy eficientemente.

Los grafos de planificación funcionan solamente en problemas de planificación proposicional, (aquellos sin variables). Como mencionamos en la Sección 11.1, tanto las representaciones STRIPS como ADL pueden ser proposicionalizadas. Para problemas con gran cantidad de objetos, esto podría convertirse en un desbordamiento del número de esquemas de acción. Dejando esto a un lado, los grafos de planificación han demostrado ser efectivas herramientas para solucionar problemas de planificación complejos.

Ilustraremos el tema de los grafos de planificación con el siguiente ejemplo (ejemplos más complejos llevarían asociados gráficos que no cabrían en la página). La Figura 11.11 nos muestra un problema, y la Figura 11.12 nos muestra su grafo de planificación.

Iniciar (*Tener*(*Pastel*))
Objetivo(*Tener*(*Pastel*) ∧ *Comido*(*Pastel*))
Acción (*Comer*(*Pastel*)
  PRECOND: *Tener*(*Pastel*)
  EFECTO: ¬*Tener*(*Pastel*) ¬ *Comido*(*Pastel*)
Acción (*Cocinar*(*Pastel*)
  PRECOND: ¬*Tener*(*Pastel*)
  EFECTO: *Tener*(*Pastel*)

Figura 11.11 El problema de «tener y comer pastel».

Figura 11.12 Grafo de planificación del problema «tener y comer pastel» del nivel S_2. Los rectángulos indican acciones (los pequeños cuadrados indican acciones persistentes) y las líneas rectas indican precondiciones y efectos. Los enlaces de exclusión mutua son representados como curvas grises.

Comenzaremos con el nivel S_0, que representa el estado inicial del problema. Continuamos con la acción A_0, en la que consideramos todas las acciones cuyas precondiciones se satisfagan en el nivel previo. Cada acción está conectada con sus precondiciones en S_0 y sus efectos en S_1, en este caso introduciendo dos nuevos literales en S_1 que no estaban en S_0.

El grafo de planificación necesita un modo de representar tanto la falta de acción como la acción. Esto es, necesita el equivalente a los axiomas del marco en el cálculo situado que permitan que un literal permanezca siendo verdad desde una situación a la siguiente si no existe acción que lo altere. En un grafo de planificación esto se logra con un conjunto de **acciones persistentes**. Para cada literal positivo y negativo C, añadimos al problema una acción persistente con la precondición C y el efecto C. La Figura 11.12 muestra una acción «real» *Comer*(*Pastel*) en A_0, junto a dos acciones persistentes dibujadas como pequeñas cajas cuadradas.

ACCIONES PERSISTENTES

El nivel A_0 contiene todas las acciones que podrían ocurrir en el estado S_0, pero sólo registra de manera importante los conflictos entre las acciones que podrían prevenirles de suceder conjuntamente. Las líneas grises en la Figura 11.12 indican enlaces de **exclusión mutua**. Por ejemplo, *Comer*(*Pastel*) se excluye mutuamente con la persistencia de *Tener*(*Pastel*) o de ¬*Comer*(*Pastel*). Veamos cómo los enlaces de exclusión mutua son computados.

EXCLUSIÓN MUTUA

El nivel S_1 contiene todos los literales que podrían resultar de escoger cualquier subconjunto de acciones en A_0. También contiene enlaces de exclusión mutua (líneas grises) que informan de los literales que podrían no aparecer juntos, pese a la elección de las acciones. Por ejemplo, *Tener*(*Pastel*) y C*omer*(*Pastel*) son mutuamente independientes; esto es, según la selección de acciones en A_0, una u otra podría ser solución, pero no ambas. En otras palabras, S_1 representa estados múltiples, al igual que una búsqueda de regresión en el espacio de estados, y los enlaces mutuamente excluyentes son restricciones que definen el conjunto de posibles estados.

Continuaremos de este modo, alternando entre el estado de nivel S_i y el nivel de acción A_i hasta que alcancemos un nivel donde dos niveles consecutivos sean idénticos. En esta situación, decimos que el grafo está **estabilizado**. Cada nivel posterior será idéntico, hasta que la expansión sea innecesaria.

ESTABILIZADO

Lo que obtenemos con esto es una estructura donde cada nivel A_i contiene todas las acciones que son aplicables en S_i, junto con restricciones que nos indican qué parejas de acciones no pueden ser ejecutadas juntas. Cada nivel S_i contiene todos los literales que podrían obtenerse de cualquier posible elección de acciones en A_{i-1}, junto con las restricciones que especifican las parejas de literales que no son posibles. Es importante hacer notar que el proceso de diseño de un grafo de planificación no requiere elección entre acciones, lo que podría llevarnos a una búsqueda combinatoria. En lugar de eso, se registra la imposibilidad de ciertas selecciones usando enlaces mutuamente excluyentes. La complejidad de construir grafos de planificación es de orden polinomial bajo, dependiente del número de acciones y literales, mientras que el espacio de estados tiene orden exponencial en términos del número de literales.

Definiremos ahora enlaces mutuamente excluyentes entre acciones y literales. Una relación mutuamente excluyente ocurre entre dos *acciones* en un nivel dado si cualquiera de las tres condiciones siguientes se cumplen:

- *Efectos inconsistentes*: una acción niega el efecto de la otra. Por ejemplo, *Comer*(*Pastel*) y la persistencia de *Tener*(*Pastel*) tienen efectos inconsistentes porque no coinciden sus efectos.
- *Interferencia*: uno de los efectos de una de las acciones es la negación de una precondición de la otra. Por ejemplo, *Comer*(*Pastel*) interfiere con la persistencia de *Tener*(*Pastel*) mediante la negación de sus precondiciones.
- *Necesidades que entran en competencia*: una de las precondiciones de una acción es mutuamente excluyente con una precondición de la otra. Por ejemplo, *Cocinar*(*Pastel*) y *Comer*(*Pastel*) son mutuamente excluyentes porque ambas compiten sobre el valor de la precondición de *Tener*(*Pastel*).

Una relación de exclusión mutua sucede entre dos *literales* en el mismo nivel, si uno es la negación del otro o si cada posible par de acciones que podrían alcanzar los literales son mutuamente excluyentes. A esta condición se le llama *soporte inconsistente*. Por ejemplo, *Tener*(*Pastel*) y C*omer*(*Pastel*) son mutuamente excluyentes en S_1 porque la única forma de alcanzar *Tener*(*Pastel*), la acción persistente, es mutuamente excluyente con la única forma de alcanzar *Comido*(*Pastel*). En S_2 los dos literales no son mutuamente excluyentes porque hay otros modos de alcanzarlos, tal como *Cocinar*(*Pastel*) y la persistencia de *Comido*(*Pastel*), que no son mutuamente excluyentes.

Grafos de planificación para estimación de heurísticas

Un grafo de planificación, una vez construido, es una fuente rica en información acerca de un problema. Por ejemplo, *un literal que no aparece en el nivel final del grafo no puede ser alcanzado por ningún plan*. Esta observación puede se usada en búsquedas hacia atrás como se muestra a continuación: cualquier estado conteniendo un literal inalcanzable tiene un coste $h(n) = \infty$. De manera similar, en planificaciones de orden parcial, cualquier plan con una condición abierta inalcanzable tiene un $h(n) = \infty$.

COSTE DE NIVEL

Esta idea puede plantearse de un modo más general. Podemos estimar el coste que supone alcanzar cualquier literal del objetivo como el nivel en el cual aparece primeramente el grafo de planificación. Lo llamaremos **coste de nivel** del objetivo. En la Figura 11.12, *Tener*(*Pastel*) tiene coste de nivel 0 y *Comido*(*Pastel*) tiene coste de nivel 1. Es sencillo mostrar (Ejercicio 11.9) que estas estimaciones son admisibles para objetivos individuales. La estimación puede nos ser muy buena, sin embargo, los grafos de planificación permiten varias acciones en cada nivel mientras la heurística compute simplemente niveles y no el número de las acciones. Por esta razón, es común usar un **grafo de planificación serial** para las heurísticas. Un grafo serial exige que sólo una acción pueda ocurrir en una etapa de tiempo dada; esto se logra añadiendo enlaces de exclusión mutua entre cada par de acciones excepto acciones persistentes. Los costes de nivel extraídos de grafos seriales son frecuentemente estimaciones razonables de costes reales.

GRAFO DE PLANIFICACIÓN SERIAL

NIVEL MÁXIMO

NIVEL SUMA

Para estimar el coste de una secuencia de objetivos, existen tres enfoques simples. La heurística de **nivel máximo** simplemente toma el coste del máximo nivel de cualquiera de los objetivos; esto es admisible pero no necesariamente preciso. La heurística de **nivel suma** (asumiendo la hipótesis de independencia de objetivos) devuelve la suma de los costes de los niveles objetivo. Esto no es admisible pero funciona bien en la práctica en problemas que son en buena parte descomponibles. Es más preciso que la aplicación de heurísticas de número de objetivos insatisfechos presentada en la Sección 11.2. En nuestro problema, la estimación heurística de la secuencia de objetivos *Tener*(*Pastel*) $\wedge$ *Comido*(*Pastel*) será $0 + 1 = 1$, mientras que la respuesta correcta es 2. Sin embargo, si eliminamos la acción *Cocinar*(*Pastel*), la estimación podría ser 1, pero la secuencia de objetivos sería imposible. Finalmente, la heurística de **nivel de conjunto** encuentra el nivel en el que todos los literales de la secuencia de objetivos aparecen en el grafo de planificación, sin que ningún par de ellos sean mutuamente excluyentes. Esta heurística nos da el valor correcto, 2, para nuestro problema original sin *Cocinar*(*Pastel*). Es mejor que la heurística de nivel máximo y además trabaja bien en tareas en las que hay buenas relaciones de interacción entre subplanes.

NIVEL DE CONJUNTO

El grafo de planificación, como herramienta de generación de heurísticas adecuadas que es, nos permite entenderlo también como un problema aproximado que es eficientemente resoluble. Para entender la naturaleza del problema aproximado, necesitamos entender exactamente qué significa que un literal g aparezca en un nivel S_i en un grafo de planificación. Idealmente, querríamos que fuese una garantía de la existencia de un plan la aparición de una acción de nivel i que alcance g, y de igual modo que si g no apareciese no existe tal plan. Desafortunadamente, obtener esa garantía es tan difícil como resolver el problema de planificación original. De manera que el grafo de planificación

simple cumple la segunda condición de la garantía (si *g* no aparece, no hay plan), pero en el caso de que *g* aparezca, entonces todo lo que el grafo de planificación asegura es que existe un plan que posiblemente alcance *g* y sin defectos «obvios». Un defecto obvio se define como un error que puede ser detectado considerando dos acciones o dos literales en un instante de tiempo (en otras palabras, observando las relaciones de exclusión mutua). Podrían existir más errores no obvios implicando a tres, cuatro o más acciones, pero la experiencia ha mostrado que no merece la pena el esfuerzo computacional que supone seguir el rastro de estos posibles errores. Es un resultado similar al que se mostró al estudiar los problemas de satisfacción de restricciones, donde era preferible computar 2-consistencias antes de la búsqueda de soluciones, pero no tan preferible como el hecho de computar 3-consistencias o más (*véase* Sección 5.2).

El algoritmo GRAPHPLAN

Esta subsección muestra cómo extraer un plan directamente de un grafo de planificación, preferiblemente al uso del plan como forma de obtener una heurística. El algoritmo GRAPHPLAN (Figura 11.13) tiene dos etapas fundamentales, las cuales se alternan dentro del ciclo del algoritmo. Primeramente, se chequea si todos los literales del objetivo están presentes en el nivel actual sin que existan enlaces mutuamente excluyentes entre cualquier par de ellos. Si éste es el caso, entonces una solución *podría* existir dentro del grafo actual, de modo que el algoritmo intentase extraer esa solución. Además, se extiende el grafo añadiendo acciones para el nivel actual y literales de estado para el siguiente nivel. El proceso continúa hasta que una solución sea encontrada o se compruebe que la solución no existe.

```
función GRAPHPLAN (problema) devuelve solución o error
  grafo ← GRAFO-PLANIFICACIÓN-INICIAL (problema)
  objetivos ← OBJETIVOS [problema]
bucle
  si objetivos todos los enlaces mutuamente excluyentes están en el último nivel del grafo
  entonces
    solución ← EXTRAER SOLUCIÓN (grafo, objetivos, LONGITUD(grafo))
    si solución ≠ error entonces devuelve solución
    en caso contrario si NO SOLUCIÓN POSIBLE (grafo) entonces devuelve error
  grafo ← GRAFO-EXPAND (grafo, problema)
```

Figura 11.13 El algoritmo GRAPHPLAN alterna entre una etapa de extracción de solución y una etapa de expansión de grafo. EXTRAER-SOLUCIÓN busca si un plan puede ser encontrado, comenzando desde el final y hacia atrás. EXPANDIR-GRAFO añade las acciones para el estado actual y los literales de estado para el nivel siguiente.

Representemos las operaciones de GRAPHPLAN en el problema de la rueda de repuesto de la Sección 11.1. El grafo completo es mostrado en la Figura 11.14. La primera línea del GRAPHPLAN inicia el grafo de planificación en un grafo de un único nivel (S_0) que

Figura 11.14 Grafo de planificación para el problema de la rueda de repuesto después de la expansión del nivel S_2. Los enlaces mutuamente excluyentes son mostrados como líneas de color gris. Solamente algunos de estos enlaces son mostrados, porque el grafo podría ser demasiado confuso en caso contrario. La solución es indicada por las líneas negras de remarcado.

consiste en cinco literales del estado inicial. El literal objetivo *En*(*Repuesto, Eje*) no está presente en S_0, de modo que no necesitamos llamar a EXTRAER-SOLUCIÓN pues estamos seguros de que no hay una solución. Por ejemplo, EXPANDIR-GRAFO añade tres acciones cuyas precondiciones existen en el nivel S_0 (esto es, todas las acciones excepto *Colocar*(*Repuesto, Eje*)), junto con las acciones persistentes para todos los literales en S_0. Los efectos de las acciones son añadidos en el nivel S_1. EXPANDIR-GRAFO busca las relaciones mutuamente excluyentes y las añade al grafo.

En(*Repuesto, Eje*) aún no está presente en S_1, así que de nuevo no llamamos a EXTRAER-SOLUCIÓN. La llamada a EXPANDIR-SOLUCIÓN nos da el grafo de planificación mostrado en la Figura 11.14. Ahora que tenemos el conjunto completo de acciones, vale la pena pasar a ver algunos ejemplos de relaciones de exclusión mutua y sus causas:

- ***Efectos inconsistentes:*** *Quitar*(*Repuesto,Maletero*) es una exclusión mutua con *DejarloSinProtecciónNoche* porque una tiene el efecto *En*(*Repuesto, Eje*) y la otra su negación.
- ***Interferencia:*** *Quitar*(*Repuesto,Maletero*) es una exclusión mutua con *DejarloSinProtecciónNoche* porque uno tiene la precondición *En*(*Pinchada, Eje*) y la otra tiene su negación como un efecto.
- ***Necesidades en competencia:*** *Colocar*(*Repuesto, Eje*) es una exclusión mutua con *Quitar*(*Pinchada, Eje*) porque uno tiene *En*(*Pinchada, Eje*) como una precondición y el otro tiene su negación.
- ***Soporte inconsistente:*** *En*(*Repuesto, Eje*) es excluyente mutuamente con *En*(*Pinchada, Eje*) en S_2 porque el único modo de alcanzar *En*(*Repuesto, Eje*) es mediante *Colocar*(*Repuesto, Eje*), y esto es excluyente mutuamente con la acción persistente que es el único modo de alcanzar *En*(*Pinchada, Eje*). De este modo, las relaciones mutuamente excluyentes detectan el conflicto inmediato que aparece al intentar colocar dos objetos en el mismo lugar y en el mismo instante de tiempo.

En el momento en el que regresamos al inicio del ciclo del algoritmo, todos los literales del objetivo están presentes en S_2, y ninguno de ellos es excluido mutuamente por otro. Esto significa que una solución podría existir, y EXTRAER-SOLUCIÓN intentaría encontrarla. En esencia, EXTRAER-SOLUCIÓN resuelve un CSP booleano cuyas variables son las acciones de cada nivel, y los valores de cada variable están dentro o fuera del plan. Podemos usar algoritmos CSP estándares para esto, o podemos definir EXTRAER-SOLUCIÓN como un problema de búsqueda, donde cada estado en la búsqueda contiene un puntero a un nivel en el grafo de planificación y un conjunto de objetivos no satisfechos. Definimos este problema de búsqueda como sigue:

- El estado inicial es el último nivel del grafo de planificación; S_n, junto con el conjunto de objetivos del problema de planificación.
- Las acciones disponibles en un estado de nivel S_i están para seleccionar cualquier subconjunto de acciones libre de conflicto en A_{i-1} cuyos efectos cubran los objetivos en el estado. El estado resultante tiene nivel S_{i-1} y tiene como conjunto de objetivos las precondiciones para el conjunto seleccionado de acciones. Por «libre de conflicto» queremos decir conjunto de acciones tales que ninguna de ellas sean mutuamente excluyentes, y que ninguna de sus precondiciones lo sean tampoco.
- El objetivo es alcanzar un estado a nivel S_0 tal que todos los objetivos sean satisfechos.
- El coste de cada acción sea 1.

Para este problema particular, comenzamos en S_2 con el objetivo *En*(*Repuesto, Eje*). La única elección posible que tenemos para alcanzar el conjunto objetivo es *Colocar*(*Repuesto, Eje*). Esto nos lleva a una búsqueda de estado en S_1 con objetivos *En*(*Repuesto, Suelo*) y no ¬*En*(*Pinchado, Eje*). El primero puede ser alcanzado sólo mediante *Quitar*(*Repuesto, Maletero*), y el último o por *Quitar*(*Pinchado, Eje*) o *DejarloSinProtecciónNoche*. Como *DejarloSinProtecciónNoche* es mutuamente excluyente con *Quitar*(*Repuesto, Maletero*), la única solución es elegir *Quitar*(*Repuesto, Maletero*) y *Quitar*(*Pinchado, Eje*). Esto nos lleva a una búsqueda de estado en S_0 con los objetivos *En*(*Repuesto, Maletero*) y *En*(*Pinchado, Eje*). Ambos están presentes en el estado, de modo que tenemos una solución: las acciones *Quitar*(*Repuesto, Maletero*) y *Quitar*(*Pinchado, Eje*) en el nivel A_0 seguido por *Colocar*(*Repuesto, Eje*) en A_1.

Conocemos que la planificación es PSPACE-completa y que el desarrollo del grafo de planificación lleva asociado un orden en tiempo polinomial, de modo que puede darse la situación en que la extracción de la solución sea intratable en el peor de los casos. Por tanto, necesitaremos alguna orientación heurística para la selección entre acciones durante la búsqueda hacia atrás. Un enfoque que trabaja bien en la práctica es un algoritmo basado en el coste de nivel de los literales. Para cualquier conjunto de objetivos, procedemos en el siguiente orden:

1. Seleccionar primero el literal con el coste de nivel más alto.
2. Para alcanzar este literal, elegir la acción con la precondición más sencilla. Esto es, elegir una acción tal que la suma (o el máximo) del nivel de coste de sus precondiciones sea la más pequeña.

Interrupción de GRAPHPLAN

Hasta ahora, hemos tratado por encima la cuestión de la interrupción. Si un problema no tiene solución ¿podemos asegurar que GRAPHPLAN no caerá en un bucle, extendiendo el grafo de planificación en cada iteración? La respuesta es afirmativa, pero la demostración completa está más allá de las pretensiones de este libro. Aquí, simplemente indicamos las ideas principales, particularmente aquellas que arrojen luz sobre los grafos de planificación en general.

El primer paso es destacar que ciertas propiedades de los grafos de planificación están creciendo o decreciendo monótonamente. Que «X crezca monótonamente» significa que el conjunto de Xs en el nivel $i + 1$, es un superconjunto (no necesariamente propio) del conjunto en el nivel i. Las propiedades son las siguientes:

- *Los literales crecen monótonamente*: una vez que un literal aparece en un nivel dado, aparecerá en todos los niveles siguientes. Esto está motivado por las acciones persistentes; una vez que un literal se pone de manifiesto, las acciones de persistencia ocasionan que permanezca indefinidamente.
- *Las acciones crecen monótonamente*: una vez que una acción aparece en un nivel dado, aparecerá en todos los niveles siguientes. Esto es consecuencia del aumento de literales; si las precondiciones de una acción aparecen en un nivel, entonces aparecerán en los niveles siguientes.
- *Los enlaces de exclusión mutua decrecen monótonamente*: si dos acciones son mutuamente excluyentes en un nivel dado A_i, entonces también lo son para todos los niveles previos en los cuales ambas aparezcan. Lo mismo sucede para exclusiones mutuas entre literales. Puede que no aparezca esta opción en las figuras porque las figuras son una simplificación: no presentan ni los literales que no pueden cumplirse en el nivel S_i ni las acciones que no pueden ser ejecutadas en el nivel A_i. Podemos ver que «los enlaces mutuamente excluyentes decrecen monótonamente» es cierto si consideramos que estos literales no son visibles y las acciones son excluyentes mutuamente con todos lo demás.

  La demostración es algo compleja, pero puede ser presentada por casos: si las acciones A y B son excluyentes mutuamente en el nivel A_i, debe ser por causa de uno de los tres tipos de exclusión mutua. Los dos primeros, efectos inconsistentes e interferencia, son propiedades de las acciones, de modo que si las acciones son mutuamente excluyentes en A_i, lo serán en cualquier nivel. El tercer caso, necesidades en competencia, depende de las condiciones de nivel S_i: ese nivel debe contener una precondición de A que sea mutuamente excluyente con una precondición de B. Ahora, esas dos precondiciones pueden ser mutuamente excluyentes si son negaciones una de otra (en cuyo caso, serían mutuamente excluyentes en cada nivel) o si todas las acciones para alcanzar una son mutuamente excluyentes con todas las acciones para alcanzar la otra. Pero ya conocemos que las acciones disponibles son crecientes monótonamente, así que por inducción, las relaciones excluyentes mutuamente deben estar decreciendo.

Como las acciones y los literales crecen y las relaciones mutuamente excluyentes decrecen, y como sólo existe un número finito de acciones y literales, cada grafo de pla-

nificación se estabiliza (todos los niveles siguientes serán idénticos). Una vez que un grafo se ha estabilizado, si le falta uno de los objetivos del problema, o si dos de los objetivos son mutuamente excluyentes, entonces el problema nunca podrá ser resuelto, y podemos detener el algoritmo GRAPHPLAN y determinar el error. Si el grafo se estabiliza con todos los objetivos y sin relaciones mutuamente excluyentes, pero EXTRAER-SOLUCIÓN fracasa al encontrar una solución, entonces tendremos que extender el grafo de nuevo un número finito de veces, aunque finalmente podremos parar. Este aspecto de interrupción es más complejo y no se mostrará aquí.

11.5 Planificación con lógica proposicional

Vimos en el Capítulo 10 que la planificación puede ser hecha mediante la demostración de un teorema del cálculo situado. Este teorema dice que, dado el estado inicial y dados los axiomas de estados sucesivos que describen los efectos de las acciones, el objetivo será correcto en una situación resultado de una cierta secuencia de acciones. A principios de 1969, se pensaba que este enfoque era demasiado ineficiente para encontrar planes interesantes. Recientes desarrollos en algoritmos de razonamiento eficiente para lógica proposicional (*véase* Capítulo 7) han generado renovado interés en la interpretación de la planificación como razonamiento lógico.

El enfoque que tomamos en esta sección está basado en la evaluación de la **satisfabilidad** de una secuencia lógica preferiblemente a la demostración de un teorema. Buscaremos modelos de sentencias proposicionales que sigan la estructura:

$$\textit{Estado inicial} \wedge \textit{todas las posibles descripciones de acción} \wedge \textit{objetivo}$$

La sentencia contendrá símbolos proposicionales correspondiéndose con cada acción posible; un modelo que satisfaga la sentencia le asignará *verdad* a las acciones que son parte de un plan correcto y *falso* a las otras. Una asignación que se corresponda a un plan incorrecto no será un modelo, porque será inconsistente con la afirmación de que el objetivo es correcto. Si el problema de planificación es irresoluble, la sentencia no será satisfecha.

Descripción de problemas de planificación en lógica proposicional

El proceso que seguiremos para traducir problemas STRIPS a una formulación en lógica proposicional es un ejemplo clásico del ciclo de representación de conocimiento: comenzaremos con lo que parece un conjunto razonable de axiomas, veremos que estos axiomas tienen en cuenta modelos espurios no buscados, y escribiremos más axiomas.

Comencemos con un problema muy simple de transporte aéreo. En el estado inicial (tiempo 0), el avión P_1 está en *SFO* y el avión P_2 en *JFK*. El objetivo es tener P_1 en *JFK* y P_2 en *SFO;* esto es, los aviones deben cambiar de aeropuerto. En primer lu-

gar, necesitamos distinguir símbolos proposicionales para afirmaciones en cada instante de tiempo. Usaremos superíndices para denotar el instante, más bien el escalón temporal, como hicimos en el Capítulo 7. De este modo, el estado inicial se expresará:

$$En(P_1, SFO)^0 \wedge En(P_2, JFK)^0$$

Recordemos que $En(P_1, SFO)^0$ es un símbolo atómico. Como la lógica proposicional no tiene una hipótesis de mundo cerrado, podemos especificar la proposiciones que *no* son correctas en el estado inicial. Si algunas proposiciones son desconocidas en el estado inicial, entonces pueden ser dejadas sin satisfacer (**hipótesis de mundo abierto**). En este ejemplo especificamos:

$$\neg En(P_1, JFK)^0 \wedge \neg En(P_2, SFO)^0$$

El objetivo debe ser asociado con un instante de tiempo particular. Dado que no conocemos a priori cuántas etapas se necesitan para alcanzar el objetivo, podemos intentar afirmar que el objetivo es correcto en el instante inicial $T = 0$. Esto es, afirmamos $En(P_1, JFK)^0 \wedge En(P_2, SFO)^0$. Si esto fracasa, lo intentaremos de nuevo en $T = 1$, y así sucesivamente hasta que alcancemos el plan más corto posible. Para cada valor de T, la base de conocimiento incluirá sólo sentencias cubriendo el intervalo temporal desde cero hasta T. Para asegurar que se interrumpe, un límite superior arbitrario, $T_{máx}$, es impuesto. Este algoritmo se muestra en la Figura 11.15. Un enfoque alternativo que evita tentativas de múltiples soluciones es discutido en el Ejercicio 11.17.

```
función SATPLAN (problema, T_máx) devuelve solución o error
  inputs : problema, problema de planificación
           T_máx, un límite superior para la longitud del plan
  para T = 0 hasta T_máx hacer
    cnf, correspondencia ← TRADUCIR A SAT (problema, T)
    asignación ← RESOLVEDOR SAT (cnf)
    si asignación no es nula entonces
    devuelve EXTRAER-SOLUCIÓN(asignación, correspondencia)
  devuelve error
```

Figura 11.15 El algoritmo SATPLAN. El problema de planificación es traducido a sentencias CNF en las que el objetivo se pretende alcanzar en un tiempo fijo T y son incluidos axiomas para cada estado previo a T. (Los detalles de la traducción son ofrecidos en el texto.) Si el algoritmo de satisfabilidad encuentra un modelo, entonces es extraído un plan a partir de símbolos proposicionales que refieran acciones y que sean *verdaderos* en el modelo. Si el modelo no existe, el proceso es repetido con el objetivo desplazado a una posición posterior.

La siguiente tarea es la de codificar descripciones de acción en lógica proposicional. El enfoque más sencillo es tener un símbolo proposicional para cada acción; por ejemplo, $Volar(P_1, SFO, JFK)^0$ es verdad si el avión P_1 vuela de *SFO* a *JFK* en el estado T = 0. Como en el Capítulo 7, escribimos versiones proposicionales de los axiomas de

estado-sucesivo desarrollado para el cálculo situado (*véase* Capítulo 10). Por ejemplo, tenemos

$$En(P_1, JFK)^1 \Leftrightarrow (En(P_1, JFK)^0 \wedge \neg(Volar(P_1, JFK, SFO)^0 \wedge En(P_1, JFK)^0)) \\ \vee (Volar(P_1, SFO, JFK)^0 \wedge En(P_1, SFO)^0). \quad (11.1)$$

Esto es, el avión P_1 estará en *JFK* a tiempo 1 si estaba en *JFK* a tiempo 0 y no ha volado a otro lugar, o estaba en *SFO* a tiempo 0 y ha volado a *JFK*. Necesitaremos uno de tales axiomas para cada avión, aeropuerto y tramo temporal. Sin embargo, cada aeropuerto adicional añade otra forma de viajar hacia (o desde) un aeropuerto dado y de este modo añade más disyunciones al término del lado derecho de cada axioma.

Con estos axiomas en juego, podemos utilizar el algoritmo de satisfabilidad para encontrar un plan. Debería ser un plan que alcanzase el objetivo a instante T = 1, a saber, el plan en el que dos aviones cambian el lugar. Ahora, supongamos que la KB es:

$$Estado\ inicial \wedge axiomas\ de\ estado\text{-}sucesivo \wedge objetivo^1 \quad (11.2)$$

lo que afirma que el objetivo es verdadero a tiempo T = 1. Se puede comprobar que un modelo de KB es la asignación en la que

$$Volar(P_1, JFK, SFO)^0 \text{ y } Volar(P_2, JFK, SFO)^0$$

son verdaderos y todos los otros símbolos de acción son falsos. Hasta aquí, todo correcto. ¿Existen otros modelos posibles que el algoritmo de satisfabilidad podría conocer? Por supuesto que sí. ¿Son todos estos modelos planes satisfactorios? No. Considérese el plan absurdo especificado por los símbolos de acción

$$Volar(P_1, SFO, JFK)^0 \text{ y } Volar(P_1, JFK, SFO)^0 \text{ y } Volar(P_2, JFK, SFO)^0$$

Este plan es absurdo porque el avión P_1 comienza en *SFO*, de modo que la acción $Volar(P_1, JFK, SFO)^0$ es inviable. Sin embargo, ¡el plan es un modelo de la sentencia de la Ecuación (11.2)!, es decir, es consistente con todo lo que hemos dicho hasta ahora acerca del problema. Para entender por qué, necesitamos mirar más cuidadosamente qué dicen los axiomas de estado sucesivos (Ecuación 11.1) acerca de las condiciones cuyas precondiciones no son satisfechas. Los axiomas predicen correctamente que nada sucederá cuando una de estas acciones sea ejecutada (Ejercicio 11.15), pero *no* dicen nada acerca de qué acciones no pueden ser ejecutadas. Para evitar la generación de planes con acciones prohibidas, debemos añadir **axiomas de precondición** estableciendo que para que una acción ocurra requiere que sus precondiciones sean satisfechas[6]. Por ejemplo, necesitamos:

AXIOMAS DE PRECONDICIÓN

$$Volar(P_1, JFK, SFO)^0 \Rightarrow En(P_1, JFK)^0$$

Como está establecido que $En(P_1, JFK)^0$ es falso en el estado inicial, este axioma asegura que cada modelo que tenga también $Volar(P_1, JFK, SFO)^0$ es falso. Añadiendo axiomas de precondición, existe un modelo que satisface todos los axiomas cuando el objetivo pretende ser alcanzado a tiempo 1, en concreto el modelo en el que el avión P_1

[6] Notemos que añadir axiomas de precondición significa que no necesitamos incluir precondiciones para la acción en axiomas de estado-sucesor.

vuela hasta *JFK* y el P_2 vuela a *SFO*. Notemos que esta solución posee dos acciones paralelas, de modo equivalente a utilizar GRAPHPLAN o POP.

Más sorpresas emergen cuando añadimos un tercer aeropuerto, *LAX*. Ahora, cada avión tiene dos acciones que son legales en cada estado. Cuando utilizamos el algoritmo de satisfabilidad, encontramos que un modelo con $Volar(P_1, SFO, JFK)^0$ y $Volar(P_2, JFK, SFO)^0$ y $Volar(P_2, JFK, LAX)^0$ satisface todos los axiomas. Esto es, el estado-sucesivo y los axiomas de precondición permiten a un avión volar hacia dos destinos a la misma vez. Las precondiciones para ambos vuelos de P_2 son satisfechas en el estado inicial; los axiomas de estado-sucesivo nos dicen que P_2 estará en *SFO* y en *LAX* a tiempo 1; por tanto el objetivo es satisfecho. Evidentemente, debemos añadir más axiomas para eliminar estas soluciones espurias. Un método es añadir **axiomas de exclusión de acciones** que prevengan acciones simultáneas. Por ejemplo, podemos insistir en la exclusión completa añadiendo todos los posibles axiomas de la forma,

AXIOMAS DE EXCLUSIÓN DE ACCIONES

$$\neg(Volar(P_2, JFK, SFO)^0 \wedge Volar(P_2, JFK, LAX)^0)$$

Estos axiomas aseguran que no pueden ocurrir dos acciones al mismo tiempo. Eliminan planes espurios, y también fuerzan a que cada plan esté totalmente ordenado. Esto hace perder la flexibilidad a los planes ordenados parcialmente; también, mediante el aumento de etapas temporales en un plan, el tiempo de computación puede extenderse.

En vez de exclusiones completas, podemos exigir sólo una exclusión parcial, esto es, descartar acciones simultáneas sólo si interfieren con otras. Las condiciones son las mismas que para las acciones excluyentes mutuamente: dos acciones no pueden ocurrir simultáneamente si una niega una precondición o el efecto de la otra. Por ejemplo, $Volar(P_2, JFK, SFO)^0$ y $Volar(P_2, JFK, LAX)^0$ no pueden ocurrir ambas, porque cada una niega la precondición de la otra; por otro lado, $Volar(P_1, SFO, JFK)^0$ y $Volar(P_2, JFK, SFO)^0$ pueden ocurrir juntas porque los dos planes no interfieren. La exclusión parcial elimina planes espurios sin exigir una ordenación completa.

Los axiomas de exclusión a veces parecen instrumentos muy contundentes. En lugar de decir que un avión no puede volar a dos aeropuertos al mismo tiempo, podemos exigir simplemente que ningún objeto pueda estar en dos lugares a la vez.

$$\forall p, x, y, t \;\; x \neq y \;\Rightarrow\; \neg(En(p, x)^t \wedge En(p, y)^t)$$

Este hecho, combinado con los axiomas de estado-sucesivo, implica que un avión no puede volar a dos aeropuertos al mismo tiempo. Hechos como este se conocen como **restricciones de estado**. En lógica proposicional, por supuesto, tenemos que escribir todas las instancias que cubren la constricción de cada estado. En el problema del aeropuerto, la restricción de estado basta para descartar todos los planes espurios. Las restricciones de estado frecuentemente son mucho más compactas que los axiomas de exclusión de acciones, pero no son siempre fáciles de derivar de la descripción STRIPS del problema.

REATRICCIONES DE ESTADO

Resumiendo, la planificación como satisfabilidad comprende la búsqueda de modelos de sentencia conteniendo el estado inicial, el objetivo, los axiomas de estado sucesivos, los axiomas de precondición, y por último la acción de exclusión de axiomas o las restricciones de estado. Puede mostrarse que esta colección de axiomas es suficiente, en el sentido de que no hay soluciones espurias. Cualquier modelo que satisfaga la sentencia

proposicional será un plan válido para el problema original, esto es, cada linealización del plan es una secuencia correcta de acciones que alcanza el objetivo.

Complejidad de codificaciones proposicionales

El principal inconveniente del enfoque proposicional es el tamaño ingente del conocimiento que se genera del problema a partir del problema de planificación original. Por ejemplo, el esquema de acción $Volar(p, a_1, a_2)$ se transforma en $T \times |Planes| \times |Aeropuertos|^2$ símbolos proposicionales diferentes. En general, el número total de símbolos de acción viene dado por $T \times |Act| \times |O|^P$, donde $|Act|$ es el número de los esquemas de acción, $|O|$ es el número de objetos en el dominio, y P es el máximo número de argumentos de cualquier esquema de acción. El número de cláusulas es aún elevado. Por ejemplo, con 10 escalones temporales, 12 aviones y 30 aeropuertos, el axioma de exclusión de acción completa tiene 583 millones de cláusulas.

Como el número de símbolos de acción es exponencial en el número de argumentos del esquema de acción, una solución podría ser el intento de reducir el número de argumentos. Podemos hacer esto tomando prestada una idea de las redes semánticas (Capítulo 10). Las redes semánticas usan sólo predicados binarios; predicados con un mayor número de argumentos serían reducidos a un conjunto de predicados binarios que describirían cada argumento por separado. Al aplicar esta idea a un símbolo de acción tal como $Volar(P_1, SFO, JFK)^0$, se obtienen tres nuevos símbolos:

$Volar_1(P_1)^0$: el avión P_1 ha volado a tiempo 0
$Volar_2(SFO)^0$: el origen del vuelo fue SFO
$Volar_3(JFK)^0$: el destino del vuelo fue JFK

DIVISIÓN DE SÍMBOLOS

Este proceso, llamado «**división de símbolos**» elimina la necesidad de un número exponencial de símbolos. Ahora sólo necesitaremos $T \times |Act| \times P \times |O|$.

La división de símbolos, por ella misma, puede reducir el número de símbolos, pero no reduce automáticamente el número de axiomas en la Base de Conocimiento. Esto es, si cada símbolo de acción en cada cláusula fuera simplemente reemplazado por el conjunto de tres símbolos, entonces el tamaño total de la Base de Conocimiento podría permanecer prácticamente equivalente. La división de símbolos realmente reduce el tamaño de la Base de Conocimiento porque algunos de los símbolos escindidos son irrelevantes a ciertas acciones y pueden ser omitidos. Por ejemplo, consideremos el axioma de estado-sucesivo en la Ecuación 11.1, modificado para incluir LAX y para omitir precondiciones de acción (las cuales son cubiertas por axiomas de precondición separados):

$$En(P_1, JFK)^1 \Leftrightarrow (En(P_1, JFK)^0 \wedge \neg Volar(P_1, JFK, SFO)^0 \wedge \neg Volar(P_1, JFK, LAX)^0) \\ \vee (Volar(P_1, SFO, JFK)^0 \vee (Volar(P_1, LAX, JFK)^0$$

La primera condición dice que P_1 estará en JFK si estuviese allí a tiempo 0 y no volase desde JFK a cualquier otra ciudad, no importa cuál. Usando símbolos divididos, podemos simplemente omitir el argumento cuyo valor no nos interesa:

$$En(P_1, JFK)^1 \Leftrightarrow (En(P_1, JFK)^0 \wedge \neg(Volar_1(P_1)^0 \wedge Volar_2(JFK)^0)) \\ \vee (Volar_1(P_1)^0 \wedge Volar_3(JFK)^0)$$

Notemos que *SFO* y *LAX* no son mencionados más en el axioma. De forma más general, la división de símbolos de acción permiten ahora que el tamaño de cada axioma de estado-sucesivo sea independiente del número de aeropuertos. Reducciones similares ocurren con los axiomas de precondición y los axiomas de exclusión de acción (*véase* Ejercicio 11.6). Para el caso descrito antes con 10 tramos temporales, 12 aviones y 30 aeropuertos, el axioma de exclusión completa es reducida de 583 millones de cláusulas a 9.360 cláusulas.

Existe un inconveniente: la representación de división-de-símbolos no permite acciones paralelas. Consideremos las dos acciones paralelas $Volar(P_1, SFO, JFK)^0$ y $Volar(P_2, JFK, SFO)^0$. Transformándolos a una representación dividida, tenemos:

$$Volar_1(P_1)^0 \wedge Volar_2(SFO)^0 \wedge Volar_3(JFK)^0 \wedge$$
$$Volar_1(P_2)^0 \wedge Volar_2(JFK)^0 \wedge Volar_3(SFO)^0$$

No es posible determinar lo que ha sucedido. Sabemos que P_1 y P_2 volaron, pero no podemos identificar el origen y el destino de cada vuelo. Esto significa que un axioma de exclusión completa debe ser usado, con el inconveniente mencionado previamente.

Los planificadores basados en la satisfabilidad pueden abordar importantes problemas de planificación, por ejemplo, encontrar soluciones óptimas de 30 etapas para problemas de planificación en mundos de bloques con docenas de bloques. El tamaño de la codificación proposicional y el coste de la solución son problemas fuertemente dependientes, sin embargo en muchos casos la memoria requerida para almacenar los axiomas proposicionales se convierte en un cuello de botella. Una interesante forma de plantear este problema ha sido mediante algoritmos de retro-propagación, tales como DPLL, que son muchas veces mejores para la solución de problemas que los algoritmos de búsquedas locales como WALKSAT. Esto es porque la mayoría de los axiomas proposicionales son cláusulas de Horn, las cuales son tratadas de manera eficiente por la unidad de propagación técnica. Esta observación ha dirigido la investigación hacia el desarrollo de algoritmos híbridos combinando búsquedas aleatorias con mecanismos de retro-propagación.

11.6 Análisis de los enfoques de planificación

La planificación es, actualmente, un área de gran interés dentro de la IA. Una razón de esto es que combina las dos mayores áreas de IA que se han tratado hasta ahora: *búsquedas* y *lógica*. Esto es, un planificador puede ser visto tanto como un programa que busca la solución o como uno que (constructivamente) proporcione la existencia de una solución. Lo fértil del cruce de ideas desde ambas áreas ha llevado a un avance en la pasada década de varios órdenes de magnitud y un aumento del uso de planificadores en aplicaciones industriales. Desafortunadamente, no tenemos una clara comprensión de qué técnicas trabajan mejor en qué clase de problemas. Muy posiblemente, emerjan nuevas técnicas que sobrepasarán a los actuales métodos.

La planificación es, en primer lugar, un ejercicio de control de la explosión combinatoria. Si tenemos un número p de proposiciones primitivas en un dominio, tendremos

2^p estados. Para dominios complejos, p puede crecer mucho. Consideremos que los objetos en el dominio tienen propiedades (*localización, color*) y relaciones (*en, sobre, entre*). Con d objetos en un dominio que mantengan relaciones cada tres, tendremos 2^{d^3} estados. Podremos concluir que, en el peor de los casos, la planificación es una tarea sin solución.

En contra de tal pesimismo, la estrategia divide-y-vencerás puede ser una poderosa arma. En el mejor caso (la descomposición completa del problema) divide-y-vencerás nos ofrece una agilización exponencial. La descomposibilidad no aporta ventajas, sin embargo, en caso de interacciones negativas entre acciones. Los planificadores de orden parcial afrontan estos problemas con enlaces causales, un poderoso enfoque de representación, pero desafortunadamente cada conflicto debe ser resuelto con una elección, y las elecciones pueden multiplicarse exponencialmente. GRAPHPLAN evita estas selecciones durante la fase de construcción del grafo, utilizando enlaces de exclusión mutua para registrar conflictos sin construir elecciones para resolverlos. SATPLAN representa un rango similar de relaciones mutuamente excluyentes, pero lo hace mediante el uso de la forma general CNF preferible a una estructura específica de datos. Lo bien o mal que vaya dependerá del solucionador SAT usado.

OBJETIVOS SERIALIZABLES

Algunas veces es posible resolver un problema eficientemente, aceptando que las interacciones negativas pueden ser descartadas. Decimos que un problema tiene **sub-objetivos serializables** si existe un orden de sub-objetivos tal que el planificador pueda alcanzarlos en ese orden, sin tener que anular ninguno de los sub-objetivos alcanzados previamente. Por ejemplo, en el mundo de bloques, si el objetivo es construir una torre (ejemplo: *A sobre B*, que se encuentra sobre *C*, que a su vez está sobre el *Tablero*), entonces los sub-objetivos son serializables de abajo-a-arriba: si primeramente situamos *C* sobre el *Tablero*, no tendremos que deshacer esta meta mientras estamos alcanzando el resto de sub-objetivos. Un planificador que usa la estrategia abajo-a-arriba puede solucionar cualquier problema en el dominio del mundo de bloques sin retro-propagación (aunque puede no encontrar siempre el mejor plan).

Presentemos un ejemplo complejo: para el agente planificador remoto que comandaba la misión espacial Deep Space One se determinó que las proposiciones vinculadas en la dirección de la misión eran serializables. Esto quizá no es demasiado sorprendente, porque una nave espacial es diseñada por ingenieros para que su control sea lo más sencillo posible (sujeto a otras restricciones). Tomando ventaja de la ordenación serializada de objetivos, el agente planificador remoto fue capaz de eliminar muchas de las búsquedas. Esto se tradujo en un control de la nave en tiempo real, algo que previamente se consideraba imposible.

Existe más de un modo de control de explosiones combinatorias. Vimos en el Capítulo 5 que hay muchas técnicas para el control de la retro-propagación en problemas que satisfagan restricciones (CSPs), tales como retro-propagación dependientemente dirigida. Todas estas técnicas pueden ser aplicadas a la planificación. Por ejemplo, extraer una solución de un grafo de planificación puede ser formulado como un problema CSP booleano cuyas variables establezcan si una acción dada debería ocurrir en un tiempo determinado. El CSP puede ser resuelto usando cualquiera de los algoritmos presentados en el Capítulo 5. Un método estrechamente relacionado, usado en sistemas BLACKBOX, convierte el grafo de planificación en una expresión CNF y posteriormen-

te extrae un plan utilizando un solucionador SAT. Este enfoque parece trabajar mejor que SATPLAN, a priori porque el grafo de planificación ha eliminado ya muchos de los estados imposibles y acciones del problema. Funciona también mejor que GRAPHPLAN, presumiblemente porque una búsqueda de satisfabilidad tal como WALKSAT tiene mucha más flexibilidad que una simple búsqueda de retro-propagación que use GRAPHPLAN.

No hay duda de que planificadores tales como GRAPHPLAN, SATPLAN y BLACKBOX han hecho progresar el campo de la planificación, tanto por la versatilidad de los sistemas de planificación como por la clarificación a nivel representacional de los temas relacionados. Estos métodos son, por tanto, inherentemente proposicionales y están limitados a los dominios donde pueden expresarse. (Por ejemplo, problemas logísticos con una pequeña docena de objetos y de localizaciones, pueden requerir gigabytes de almacenamiento para las correspondientes expresiones CNF.) Parece probable que se requieran representaciones de primer orden y nuevos algoritmos que acompañen el progreso en el área, mientras las estructuras tales como grafos de planificación continuarán siendo útiles como fuente de heurísticas.

11.7 Resumen

En este capítulo, hemos definido el problema de planificación en entornos deterministas, y completamente observables. Describimos las representaciones principales usadas para problemas de planificación y varias estrategias algorítmicas para resolverlos. Los elementos que tenemos que recordar son:

- Los sistemas de planificación son algoritmos de resolución de problemas que operan con representaciones proposicionales explícitas (de primer orden) de estado y acciones. Estas representaciones hacen posible la obtención de heurísticas efectivas y el desarrollo de poderosos y flexibles algoritmos para la resolución de problemas.
- El lenguaje STRIPS describe acciones en términos de sus precondiciones y efectos y describe el estado inicial y objetivo como secuencias de conjunciones entre literales positivos. El lenguaje ADL relaja algunas de estas restricciones, permitiendo la disyunción, la negación y los cuantificadores.
- Las búsquedas en el espacio de estados pueden operar hacia delante (**progresión**) o hacia atrás (**regresión**). Heurísticas efectivas pueden obtenerse aceptando la hipótesis de sub-objetivos independientes y mediante varias aproximaciones del problema de planificación.
- Los algoritmos de Planificación de Orden Parcial (POP) exploran el espacio de planes sin comprometerse con una secuencia de acciones totalmente ordenada. Trabajando hacia atrás desde el objetivo y añadiendo acciones para planificar cómo alcanzar cada sub-objetivo. Es particularmente efectivo en problemas en los que se puede utilizar una estrategia divide-y-vencerás.
- Un **grafo de planificación** puede ser construido de manera incremental, partiendo del estado inicial. Cada capa contiene un súper-conjunto de todos los literales

que podrían ocurrir en cada tramo temporal y codifica las **relaciones de exclusión mutua**, las relaciones entre literales o entre acciones que no puedan darse a la vez. Los grafos de planificación nos proporcionan útiles heurísticas en el espacio de estados, para planificadores de orden parcial, y pueden ser usados directamente en el algoritmo GRAPHPLAN.

- El algoritmo GRAPHPLAN procesa el grafo de planificación, usando una búsqueda hacia atrás hasta extraer un plan. Esto tiene en cuenta ordenaciones parciales entre acciones.
- El algoritmo SATPLAN traduce un problema de planificación en axiomas proposicionales y aplica un algoritmo de satisfabilidad para encontrar un modelo que se corresponda con un plan válido. Diferentes representaciones proposicionales han sido desarrolladas variando el grado de concisión y eficiencia.
- Cada una de las estrategias de planificación tiene sus adeptos y no existen consensos sobre cuál es la mejor. La competencia entre planteamientos y lo fértil del cruce y combinación de diferentes enfoques se ha traducido en ganancias significativas en eficiencia para sistemas de planificación.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

La planificación en IA emerge de la investigación en áreas, como búsquedas en el espacio de estados, demostración de teoremas y teoría de control, y desde las necesidades prácticas en robótica, organización y otros dominios. STRIPS (Fikes y Nilsson, 1971), el primero de los grandes sistemas de planificación, ilustra la interacción de dichas influencias. STRIPS fue diseñado como un componente para la planificación *software* del robot Shakey proyectado en el SRI. Su estructura de control total fue modelada en GPS *General Problem Solver* (Newell y Simon, 1961), un sistema de búsqueda en el espacio de estados que utilizaba un mecanismo de análisis de fines/medios. STRIPS usaba una versión del sistema de demostración de teoremas QA3 (Green, 1969b) como una subrutina para establecer la verdad de las precondiciones de las acciones. Lifschitz (1986) ofrece precisas definiciones y un análisis del lenguaje STRIPS. Bylander (1992) muestra simples planificaciones STRIPS para PSPACE-completos. Fikes y Nilsson (1993) nos ofrecen una retrospectiva histórica del proyecto STRIPS y una revisión de sus relaciones con los esfuerzos más recientes en planificación.

La representación de la acción usada por STRIPS ha sido más influyente que su estrategia algorítmica. Casi todos lo sistemas de planificación desde entonces han usado una variante u otra del lenguaje STRIPS. Desafortunadamente, la proliferación de variantes ha hecho las comparaciones necesariamente difíciles. Una mejor comprensión de las limitaciones e intercambios entre formalismos necesitaría más tiempo. El Lenguaje de Descripción de Acciones, ADL, (Pednault, 1986) relajó algunas de las restricciones en el lenguaje STRIPS e hizo posible tratar problemas más complejos. Nebel (2000) explora esquemas para compilar ADL en STRIPS. El problema del Lenguaje de Descripción de dominios o PDL (Ghallab *et al.,* 1998) fue introducido como un sintaxis estandarizada para la representación de STRIPS, ADL y otros lenguajes. PDDL ha sido usado como el

lenguaje estándar para las competiciones de planificación en la conferencia AIPS, iniciadas el año 1998.

Los planificadores, a principios de la década de los 70, generalmente trabajaban con secuencias de acciones ordenadas. La descomposición de problemas fue alcanzada por la computación de un sub-plan para cada sub-objetivo y encadenando los sub-planes en algún orden. Este enfoque, llamado por Sacerdoti **planificación lineal** (1975), pronto se demostró que era incompleto. No podía resolver algunos problemas muy simples, tales como la anomalía de Sussman (*véase* Ejercicio 11.11), planteado por Allen Brown durante su experimentación con el sistema HACKER (Sussman, 1975). Un planificador completo debe permitir **intercalar** acciones de diferentes sub-planes dentro de una simple secuencia. La noción de sub-objetivos serializables (Korf, 1987) se corresponde exactamente con el conjunto de problemas para los que los planificadores no-intercalables son completos.

PLANIFICACIÓN LINEAL

INTERCALABLES

Una solución para problemas intercalados fue la planificación de regresión de objetivos, una técnica con la que se reordenan las fases de un plan totalmente ordenado, con el fin de evitar conflictos entre sub-objetivos. Esto fue introducido por Waldinger (1975) y también usado por el WARPLAN de Warren (1974). WARPLAN es también destacable porque fue el primer planificador en ser descrito un lenguaje de programación lógico (Prolog) y es uno de los mejores ejemplos de las posibilidades que pueden algún día abrirse con el uso de la programación lógica: WARPLAN tiene solamente 100 líneas de código, una pequeña fracción de tamaño en tiempo comparable con otros planificadores. INTERPLAN (Tate, 1975a, 1975b) también permitió el intercalado arbitrario de etapas de un plan para compensar la anomalía de Sussman y los problemas relacionados.

La idea sobre las que se fundamenta que la planificación de orden parcial incluye la detección de conflictos (Tate, 1975a), y la protección de condiciones alcanzadas desde la interferencia (Sussman, 1975). La construcción de planes parcialmente ordenados (en su momento llamados **redes de tareas**) fue precursora del planificador NOAH (Sacerdoti, 1975, 1977) y el sistema NONLIN de Tate (1975b, 1977)[7].

La planificación de orden parcial dominó los 20 años siguientes de investigación, y durante gran parte de este tiempo, la aportaciones a esta área no fueron del todo entendidas. TWEAK (Chapman, 1987) fue una reconstrucción lógica y una simplificación del trabajo en planificación durante este tiempo; su formulación fue suficientemente clara para permitir demostraciones de completitud e intratabilidad (NP-completitud e indecibilidad) o diversas formulaciones del problema de planificación. El trabajo de Chapman nos lleva hasta la que fue posiblemente la primera descripción legible de un planificador de orden parcial completo (McAllester y Rosenblitt, 1991). Una implementación del algoritmo de McAllester y Rosenblitt llamado SNLP (Soderland y Weld, 1991) fue ampliamente distribuido y permitió a muchos investigadores entender y experimentar con planificación de orden parcial por primera vez. El algoritmo POP descrito en este capítulo está basado en SNLP.

---

[7] Puede existir alguna confusión con la terminología. Muchos autores usan el término **no-lineal** para referirse a parcialmente ordenado. Esto es significativamente diferente del uso original de Sacerdoti para referirse a planes intercambiables.

El grupo de Weld también desarrolló UCPOP (Penberthy y Weld, 1992), el primer planificador para problemas expresados en ADL. UCPOP incorporó la heurística de número de objetivos insatisfechos. Por primera vez se obtuvo un planificador más rápido que SNLP, pero casi nunca fue capaz de encontrar planes con más de una docena de pasos. Aunque heurísticas mejoradas fueron desarrolladas por UCPOP (Joslin y Pollack, 1994; Gereveni y Schubert, 1996), la planificación de orden parcial se vio interrumpida en la década de los 90 cuando emergieron métodos más rápidos. Nguyen y Kambhampati (2001) sugirieron que una renovación era necesaria: con heurísticas adecuadas extraídas de un grafo de planificación, y desarrollaron el planificador REPOP que avanzaba mucho mejor que GRAPHPLAN y era competitivo frente a planificadores de espacio de estados más rápidos.

Avrim Blum y Merrick Furst (1995, 1997) revitalizaron el campo de la planificación con su sistema GRAPHPLAN, que fue más rápido en varios órdenes de magnitud que los planificadores de orden parcial. Otros sistemas de planificación de grafos, tales como IPP (Koehler *et al*., 1997), STAN (Fox y Long, 1998) y SGP (Weld *et al*. 1998), pronto se pondrían en práctica. Una estructura de datos estrechamente relacionada con el grafo de planificación ha sido desarrollado hace relativamente poco por Ghallab y Laruelle (1994), cuyo planificador de orden parcial IXTET se usa para derivar adecuadas heurísticas que guíen las búsquedas. Nuestra discusión sobre grafos de planificación está basada parcialmente en este trabajo y en los trabajos de Subbarao Kambhampati. Como hemos mencionado en este capítulo, un grafo de planificación puede ser usado de muchas formas diferentes para guiar la búsqueda de una solución. El vencedor de la competición de planificación AIPS en la edición de 2002, LPG (Gerevini y Serina, 2002), buscaba grafos de planificación usando una técnica de búsqueda local inspirada por WALKSAT.

La planificación como satisfabilidad y el algoritmo SATPLAN fueron propuestos por Kautz y Selman (1992), que estuvieron inspirados por el sorprendente éxito de las búsquedas locales para la satisfacción de problemas (*véase* Capítulo 7). Kautz *et al*. (1996) también investigaron diferentes formas de representaciones proposicionales utilizando axiomas STRIPS, llegando al resultado de que las formas más compactas necesariamente no llevaban a las soluciones más satisfactorias a tiempo. Un análisis sistemático fue realizado por Ernst *et al*. (1997), quien también desarrolló un «compilador» automático para la generación de representaciones proposicionales de problemas PDL. El planificador BLACKBOX, que combina ideas de GRAPHPLAN y SATPLAN, fue desarrollado por Kautz y Selman (1998).

El resurgimiento de interés por la planificación en espacio de estados fue promovido por Drew McDermott y su programa UNPOP (1996), y fue el primero en sugerir una distancia heurística basada en la aproximación de un problema con la eliminación de listas ignoradas. El nombre UNPOP fue una reacción a la insoportable concentración de planificadores de orden parcial; McDermott sospechaba que no se le había prestado la suficiente atención que merecían otros enfoques. El planificador de búsqueda heurística de Bonet y Geffner (HSP) y sus posteriores variaciones (Bonet y Geffner, 1999) fueron los primeras en plantear búsquedas prácticas en espacio de estados para problemas de planificación complejos. El buscador de espacio de estados más exitoso en la actualidad es FASTFORWARD o FF (2000) de Hoffmann, ganador en la competición de plani-

ficación AIPS 2000. FF usa una heurística de planificación de grafos simplificada con un algoritmo de búsqueda muy rápido que combina búsquedas locales y hacia delante, de un modo novedoso.

DIAGRAMAS DE DECISIÓN BINARIA

Más recientemente, ha habido interés en la representación de planes como **diagramas de decisión binaria**, una descripción compacta de autómatas finitos ampliamente estudiada en la comunidad *hardware* (Clarke y Grumberg, 1987; McMillan, 1993). Existen técnicas para proporcionar propiedades de diagramas de decisión binaria, incluyendo la propiedad de ser una solución a un problema de planificación. Cimatti *et al*. (1998) presenta un planificador basado en este enfoque. Otras representaciones también han sido usadas; por ejemplo, Vossen *et al*. (2001) examinan el uso de la programación entera para la planificación.

Existen, aunque no de manera definitiva, algunas nuevas e interesantes comparaciones entre los diferentes enfoques de planificación. Helmert (2001) analiza varias clases de problemas de planificación, y muestra que el enfoque basado en restricciones, tal como GRAPHPLAN y SATPLAN son los mejores para dominios NP-duros, mientras que enfoques basados en búsquedas funcionan mejor en dominios donde soluciones factibles puedan ser encontradas sin técnicas de retro-propagación. GRAPHPLAN y SATPLAN tienen problemas en dominios con varios objetos, porque pueden crear varias acciones. En algunos casos, el problema puede ser aplazado o evitado por la generación dinámica de acciones proposicionalizadas preferiblemente que mediante la instanciación antes de que la búsqueda comience.

Weld (1994, 1999) proporciona dos excelentes revisiones de los algoritmos de planificación modernos. Es interesante ver el cambio que se produce entre los cinco años que pasan entre ambos enfoques: el primero se concentra en planificación de orden parcial, y el segundo introduce GRAPHPLAN y SATPLAN. *Readings in Planning* (Allen *et al.,* 1990) es una antología exhaustiva de muchos de los mejores artículos recientes en el área, incluyendo buenas revisiones. Yang (1997) proporciona una perspectiva general de técnicas de planificación de orden parcial.

La planificación ha sido central en la IA desde su inicio, y los artículos sobre planificación son un ingrediente de las principales revistas especializadas y conferencias. Existen conferencias especializadas tales como la Conferencia Internacional de Sistemas de Planificación en IA (AIPS), el Workshop Internacional en Planificación y Organización Espacial, y la Conferencia Europea sobre Planificación.

EJERCICIOS

11.1 Describa las diferencias y similitudes entre la resolución de problemas y la planificación.

11.2 Dados los axiomas de la Figura 11.2 ¿Cuáles son las instancias concretas aplicables a *Volar*(*p, desde, hasta*) en el estado descrito por

$$En(P_1, JFK) \wedge En(P_2, SFO) \wedge Avión\ (P_1) \wedge Avión\ (P_2)$$
$$\wedge\ Aeropuerto\ (JFK) \wedge Aeropuerto\ (SFO)\ ?$$

11.3 Consideremos cómo podríamos traducir un conjunto de esquemas STRIPS en un conjunto de axiomas de cálculo situado y de estado-sucesivo (*véase* Capítulo 10).

- Considérese el esquema para *Volar*(*p, desde, hasta*). Escriba una definición lógica para el predicado *VolarPrecondición*(*p, desde, hasta, s*), lo cual es correcto si las precondiciones para *Volar*(*p, desde, hasta*) son satisfechas en la situación *s*.
- En segundo lugar, y asumiendo que *Volar*(*p, desde, hasta*) es el único esquema de acción disponible al agente, escriba un axioma de estado-sucesivo para *En*(*p, x, s*) que capture la misma información que el esquema de acción.
- Ahora supongamos que existe un método de viaje adicional: *Transporte*(*p, desde, hasta*). Se tiene la precondición adicional ¬*Modificado*(*p*) y el efecto adicional *Modificado*(*p*). Explique cómo debe ser modificada la base de conocimiento de cálculo situado.
- Finalmente, desarrolle un procedimiento general y específico precisamente para llevar a cabo el traslado desde un conjunto de esquemas STRIPS a un conjunto de axiomas de estado sucesivo.

11.4 El problema del mono y los plátanos viene representado por un mono en un laboratorio con algunas bananas colgadas del techo fuera de su alcance. Una caja se encuentra disponible y le capacitaría para alcanzar las bananas si el mono se sube encima. Inicialmente, el mono está sobre *A*, las bananas sobre *B,* y la caja en *C*. El mono y la caja tiene peso *Poco*, pero si el mono se sube a la caja, el conjunto adquiere un peso *Elevado,* de igual valor que las bananas. Las acciones disponibles para el mono incluyen *Ir* de un lado a otro, *Empujar* un objeto de un lugar a otro, S*ubirse o Bajarse* de un objeto, y *Agarrar o Soltar* un objeto. *Agarrar* es el resultado de la manipulación de un objeto si el mono y el objeto están en el mismo lugar con el mismo peso.

a) Escriba la descripción del estado inicial.
b) Escriba las definiciones, en forma STRIPS, de las seis acciones.
c) Supongamos que el mono quiere confundir al científico, que ha salido a tomar el té, agarrando las bananas pero colocando de nuevo la caja en su sitio inicial. Escriba este esquema como un objetivo general (es decir, sin asumir que la caja está necesariamente en *C*) en lenguaje de cálculo situado. ¿Podría este objetivo ser alcanzado por un sistema de tipo STRIPS?
d) El axioma para *Empujar* es probablemente incorrecto, porque si el objeto es demasiado pesado, su posición permanecerá siendo la misma aún cuando el operador *Empujar* sea aplicado. ¿Es este un ejemplo de problema de ramificación o de cualificación? Asegúrese de que la descripción del problema sirva para objetos pesados.

11.5 Explique por qué el proceso para la generación de precedentes en búsquedas hacia-atrás no necesita añadir literales que representen los efectos negativos de la acción.

11.6 Explique por qué al eliminar los efectos negativos de cada esquema de acción en un problema STRIPS, se obtiene un problema aproximado.

11.7 Examine la definición de **búsqueda bidireccional** del Capítulo 3.

a) ¿Podría ser una búsqueda bidireccional en el espacio de estados una buena idea para la planificación?
b) ¿Y una búsqueda bidireccional en el espacio de planes de orden parcial?
c) Invente una versión de planificación de orden parcial en el que una acción pueda ser añadida a un plan si sus precondiciones pueden ser alcanzadas por los efectos de acciones ya existentes en el plan. Explique cómo tratar con conflictos y ordenamiento de restricciones. ¿Es el algoritmo esencialmente idéntico a una búsqueda hacia-delante en el espacio de estados?
d) Consideremos un planificador de orden parcial que combine el método de la parte (c) con el método estándar de añadir acciones para alcanzar condiciones abiertas. ¿Podría ser el algoritmo resultante el mismo que en la parte (b)?

11.8 Construya niveles 0,1 y 2 del grafo de planificación para el problema de la Figura 11.2.

11.9 Demuestre las siguientes afirmaciones acerca de los grafos de planificación:

- Un literal que no aparece en el nivel final del grafo no puede ser alcanzado.
- El coste de nivel de un literal en un grafo serial no es mayor que el coste real de un plan óptimo para alcanzarlo.

11.10 Contrastamos planificadores de búsqueda hacia atrás y hacia delante en el espacio de estados frente a planificadores de orden parcial, diciendo que los últimos son buscadores en el espacio de planes. Explique cómo búsquedas hacia delante y hacia atrás en el espacio de estados pueden también ser consideradas como busquedas en el espacio de planes, y diga cuáles son los operadores de refinamiento de planes.

11.11 La Figura 11.16 muestra un problema en un mundo de bloques conocido como la **anomalía de Sussman.** El problema fue considerado anómalo porque los planificadores que no permiten intercalar fases, de principios de los 70, no podían resolverlo. Escriba una definición del problema en notación STRIPS y resuélvalo, tanto a mano como con un programa de planificación. Un planificador no-intercalador es un planificador que, dados dos sub-objetivos G_1 *y* G_2, o produce un plan para G_1 concatenado con un

Figura 11.16 El problema de planificación «anomalía de Sussman» en un mundo de bloques.

plan para G_2, o viceversa. Explique por qué no se puede resolver este problema con un planificador de este tipo.

11.12 Consideremos el problema de ponerse uno mismo zapatos y calcetines, tal como definimos en la Sección 11.3. Aplique GRAPHPLAN a este problema y muestre la solución obtenida. Ahora añada acciones para ponerse un abrigo y un sombrero. Muestre el plan de orden parcial que es solución, y muestre que existen 180 linealizaciones diferentes del plan de orden parcial. ¿Cuál es el mínimo número de grafos de planificación diferentes y necesarios para representar las 180 linealizaciones?

11.13 El programa original STRIPS fue diseñado para controlar al robot Shakey. La Figura 11.17 muestra una versión del mundo de Shakey que consiste en cuatro habitaciones alineadas a lo largo de un pasillo; cada habitación tiene una puerta y un interruptor de luz.

Figura 11.17 El mundo de Shakey. Shakey puede moverse entre las paredes dentro de una habitación, puede pasar a través de la puerta entre dos habitaciones, puede subir a objetos escalables, empujar objetos móviles y conmutar interruptores de luz.

Las acciones en el mundo de Shakey incluyen: moverse de un sitio a otro, empujar objetos móviles (tales como cajas), escalar y bajar de objetos rígidos (tales como cajas), y encender y apagar las luces. El robot por sí mismo nunca fue suficientemente hábil para subirse sobre una caja o conmutar el interruptor para que la luz pasase de encendida a apagada y al contrario, pero el planificador STRIPS fue capaz de encontrar e imprimir planes que iban más allá de las habilidades del robot. Las seis acciones de Shakey eran las siguientes:

- *Ir*(*x, y*), que requiere que Shakey esté en *x*, además de que *x* e *y* estén localizadas en la misma habitación. Por convención, una puerta entre dos habitaciones está en ambas.
- Empujar una caja *b* desde el lugar *x* al lugar *y* dentro de la misma habitación: *Empujar* (*b, x, y*). Necesitaremos el predicado *Caja* y constantes para las cajas.
- Subirse a una caja: *Subir*(*b*); Bajar de una caja: *Bajar*(*b*). Necesitaremos el predicado *Sobre* y la constante *Suelo*.
- Encender y apagar una luz: *Encender*(*s*), Apagar(s); para encender o apagar la luz, Shakey debe estar sobre una caja que se encuentre en la posición del interruptor.

Describa las seis acciones de Shakey y el estado inicial de la Figura 11.17 en notación STRIPS. Construya un plan para Shakey que logre que la $Caja_2$ esté en la $Habitación_2$.

11.14 Vimos que los grafos de planificación solamente pueden trabajar con acciones proposicionales. ¿Qué ocurre si queremos usar grafos de planificación para un problema con variables en el objetivo, tal como $En(P_1, x) \wedge En\ (P_2,x)$, donde *x* cubre el rango sobre un dominio finito de localizaciones? ¿Cómo podría codificarse tal problema para trabajar con grafos de planificación? (Recuerde la acción *Finalizar* del planificador POP. ¿Qué precondiciones debe tener?)

11.15 Hasta ahora hemos asumido que las acciones son sólo ejecutadas en situaciones apropiadas. Veamos lo que tienen que decir axiomas proposicionales de estado sucesivo, como en la Ecuación 11.1 acerca de acciones cuyas precondiciones no son satisfechas.

a) Muestre que los axiomas predicen que no sucederá nada cuando una acción sea ejecutada en un estado donde sus precondiciones no son satisfechas.

b) Considérese un plan *p* que contiene las acciones requeridas para alcanzar un objetivo pero que también incluye acciones no permitidas. ¿Es éste un caso como

$$\textit{Estado inicial} \wedge \textit{axiomas de estado sucesivo} \wedge p \models \textit{objetivo?}$$

c) Con axiomas de estado sucesor y de primer-orden en cálculo situado (como en el Capítulo 10), ¿es posible probar que un plan que contenga acciones ilegales alcanzará el objetivo?

11.16 Utilice el dominio del problema del aeropuerto, para dar ejemplos y explique cómo la división de símbolos reduce el tamaño de los axiomas de precondición y los axiomas de exclusión de acciones. Derive una fórmula general para el tamaño de cada conjunto de axiomas en términos del número de tramos temporales, el número de esquemas de acción y el número de objetos.

11.17 En el algoritmo SATPLAN de la Figura 11.15, cada llamada al algoritmo de satisfabilidad afirma un objetivo g^T, donde T cubre el rango de $T = 0$ a $T = T_{máx}$. Suponga, en lugar de esto, que el algoritmo de satisfabilidad es llamado sólo una vez, con el objetivo $g^0 \vee g^1 \vee ... \vee g^{T_{máx}}$.

a) ¿Devuelve siempre un plan si existe uno con longitud menor o igual que $T_{máx}$?

b) ¿Este enfoque introduce alguna nueva «solución» espuria?

c) Discuta cómo se podría modificar un algoritmo de satisfabilidad tal como WALKSAT de modo que encontrase soluciones cortas (si existen) cuando se dé un objetivo disyuntivo de la forma indicada.

12 Planificación y acción en el mundo real

Donde veremos cómo representaciones más expresivas y arquitecturas de agente más interactivas nos llevan a planificadores más útiles en el mundo real.

En el capítulo anterior introdujimos la mayoría de los conceptos básicos, representaciones y algoritmos para la planificación. Los planificadores que se usan en el mundo real, por ejemplo, para la planificación de las observaciones del Telescopio Espacial Hubble, factorización operativa, en el tratamiento con problemas de logística en campañas militares son más complejos; es necesario extender los planteamientos presentados tanto en el ámbito del lenguaje de representación como en el modo en el que el planificador interactúa con el entorno. Este capítulo muestra cómo. La Sección 12.1 describe la planificación y la programación en contextos con limitación de recursos y tiempo, y la Sección 12.2 describe la planificación con subplanes predefinidos. Las Secciones 12.3 a 12.6 presentan diferentes arquitecturas de agentes diseñadas para trabajar en entornos con incertidumbre. La Sección 12.7 muestra cómo planificar cuando el entorno contiene otros agentes.

12.1 Tiempo, planificación y recursos

La representación STRIPS nos indica *qué* acciones hacer, pero, dado que la representación está basada en cálculo de situaciones, no puede decirnos nada acerca de *cuánto* dura una acción, ni acerca de *cuándo* la acción ocurre, excepto para informar que ocurre antes o después de otra de las acciones. En algunos dominios es importante conocer cuándo empiezan y terminan las acciones. Por ejemplo, en el problema del reparto de carga,

nos interesa conocer cuándo el avión que transporta la carga aterrizará, y no simplemente que el aterrizaje ocurrirá al terminar su vuelo.

PROGRAMACIÓN DE ACTIVIDADES

El tiempo está en la esencia de la familia general de ciertas aplicaciones que llamaremos **programación de actividades**. Dichos problemas requieren completar un conjunto de tareas, cada una de las cuales consiste en una secuencia de acciones, y donde cada acción tiene una duración determinada y puede requerir varios recursos. El problema está en determinar un plan que minimice el tiempo total requerido para completar todas las tareas, a la vez que se respetan las restricciones impuestas a sus recursos.

Un ejemplo de un problema de programación de actividades viene dado en la Figura 12.1. Éste es un ejemplo simplificado del problema del ensamblaje de automóviles. Tenemos dos tareas: ensamblar el coche C_1, y el coche C_2. Cada tarea consiste en tres acciones: montar el motor, las ruedas e inspeccionar los resultados. El motor debe ser el primero en ser montado (porque colocar previamente las ruedas podría reducir el espacio disponible) y, obviamente, la inspección debe hacerse en último lugar.

Iniciar ($Chasis(C_1) \wedge Chasis(C_2)$
$\wedge\ Motor(E_1, C_1, 30) \wedge Motor(E_2, C_2, 60)$
$\wedge\ Ruedas(W_1, C_1, 30) \wedge Ruedas(W_2, C_2, 15)$)
Objetivo ($Montado(C_1) \wedge Montado(C_2)$)

Acción (*AñadirMotor*(e, c, m)
PRECOND: $Motor(e, c, d) \wedge Chasis(c) \wedge \neg MotorDentro(c)$,
EFECTO: $MotorDentro(c) \wedge Duración(d)$)
Acción (*AñadirRuedas*(w, c),
PRECOND: $Ruedas(w, c, d) \wedge Chasis(c)$,
EFECTO: $RuedasPuestas(c) \wedge Duración(d)$)
Acción (*Inspeccionar*(c),
PRECOND: $MotorDentro(c) \wedge RuedasPuestas(c) \wedge Chasis(c)$,
EFECTO: $Montado(c) \wedge Duración(10)$)

Figura 12.1 Un problema de programación de actividades para el ensamblaje de dos coches. La notación *Duración*(d) significa que una acción toma d minutos en ejecutarse. $Motor(E_1, C_1, 60)$ significa que E_1 es un motor que encaja en el chasis de C_1 y que lleva 60 minutos ser instalado.

El problema de la Figura 12.1. puede ser resuelto por cualquiera de los planificadores que ya se han visto. La Figura 12.2 (ignorando los números del esquema) muestra la solución que un planificador de primer orden POP podría obtener. Interpretarlo como un problema de programación de tareas, en lugar de como un problema de *planificación*, exige determinar cuándo deben iniciarse y concluirse cada una de las acciones. Esto significa que se debe presentar atención a la duración de cada acción, tanto como a su orden. La notación *Duración*(d) como efecto de una acción (donde d debe ser limitado a un número) significa que la acción toma d minutos en ser completada.

MÉTODO DEL CAMINO CRÍTICO

Dada un ordenación parcial de acciones con la especificación de duraciones, como vemos en la Figura 12.2, podemos utilizar el **método del camino crítico** (MCC) para determinar los posibles tiempos de inicio y finalización de cada una de las acciones. Un

Figura 12.2 Solución al problema de planificación de actividades de la Figura 12.1. En la parte superior, la solución se da como un plan de orden parcial. La duración de cada acción se presenta en la parte baja de cada rectángulo, con los extremos temporales del intervalo [*ES, LS*]. La diferencia entre ambos valores es el intervalo de relajación de una acción; las acciones que presentan un valor nulo están en el camino crítico, y se muestra con flechas oscuras. En la parte inferior de la figura, la misma solución se muestra sobre un eje temporal. Los rectángulos grises representan intervalos de tiempo durante los cuales una acción puede ser ejecutada, considerando que las restricciones son respetadas. La zona desocupada en cada rectángulo gris indica el intervalo de relajación.

camino de un plan parcialmente-ordenado es una secuencia de acciones linealmente ordenadas comenzando con *Inicio* y terminando con *Final*. (Por ejemplo, existen dos caminos en el plan de orden parcial de la Figura 12.2.)

CAMINO CRÍTICO

El **camino crítico** es aquel que presenta la mayor duración; se le llama camino «crítico» porque determina la duración de un plan completo (reducir otros caminos no se traduce en un acortamiento del plan como conjunto, pero retrasar el inicio de cualquier acción del camino crítico ralentiza el plan en su conjunto). En la figura, el camino crítico se destaca con líneas gruesas de color negro. Para completar el plan completo en el mínimo tiempo total, las acciones del camino crítico deben ser ejecutadas sin retrasos entre ellas. Las acciones que quedan fuera del camino crítico poseen un margen de libertad (una ventana temporal) para ser ejecutados. Esta ventana queda especificada en términos de un intervalo de tiempo previo al inicio, *ES*, y un intervalo de tiempo posterior al inicio, *LS*. La cantidad $LS - ES$ se conoce como el intervalo de **relajación** de una acción. Podemos ver en la Figura 12.2 cómo el plan en su conjunto toma 80 minutos, que cada acción del camino crítico tiene un intervalo de relajación nulo (siempre ocu-

RELAJACIÓN

rre así) y que cada una de las acciones en el ensamblaje de C_1 tiene una ventana de 10 minutos de intervalo para que se inicie la acción. Junto con los tiempos *ES* y *LS* asociados a cada acción, se constituye el **programa** del problema.

PROGRAMA

Las siguientes fórmulas nos sirven como una definición de *ES* y *LS* y como el esquema de un algoritmo de programación para computar estos parámetros:

$$ES(Inicio) = 0$$
$$ES(B) = \text{máx}_{A < B}\, ES(A) + Duración(A)$$
$$LS(Finalizar) = ES(Finalizar)$$
$$LS(A) = \text{mín}_{A < B}\, LS(B) - Duración(A)$$

La idea es que comencemos por asignarle a *ES*(*Iniciar*) el valor 0. Entonces, tan pronto como obtengamos una acción *B* tal que todas las acciones que ocurran inmediatamente después de *B* tengan los valores de *ES* asignados, fijamos que *ES*(*B*) sea el máximo de los tiempos de finalización de las acciones inmediatamente precedentes. Este proceso se repite hasta que cada uno tenga asignado un valor *ES*. Los valores *LS* son calculados de una manera similar, hacia atrás desde la acción *Finalizar.* Los detalles se dejan como ejercicio.

El orden de complejidad de un algoritmo de camino crítico es solamente $O(Nb)$, donde *N* es el número de acciones y *b* es el máximo factor de ramificación dentro o fuera de una acción. (Para ver esto, destacamos que los cálculos de *ES* y *LS* se realizan una vez por cada acción, y que cada cálculo itera sobre un máximo de *b* acciones diferentes.) Por tanto, el problema de encontrar un programa de duración mínima, dada una ordenación parcial de las acciones, es relativamente sencillo de resolver.

Programación con restricción de recursos

RECURSOS

Los problemas de programación real son complicados por la presencia de restricciones sobre los **recursos.** Por ejemplo, añadir un motor a un coche requiere un elevador. Si sólo existe un elevador, no podremos añadir simultáneamente un motor E_1 a un coche C_1 y un motor E_2 a un coche C_2; de este modo, el programa de tareas mostrado en la Figura 12.2 podría ser inviable. El elevador del motor es un ejemplo de un **recurso reutilizable** (un recurso que se encuentra «ocupado» durante la acción pero que se vuelve disponible cuando la acción ha finalizado). Destaquemos que los recursos reutilizables no pueden ser manejados con nuestras descripciones de las acciones en términos de precondiciones y efectos, porque la cantidad de recursos disponibles no cambia después de que la acción sea completada[1]. Por esta razón, incrementamos nuestra representación para incluir un campo de la forma RECURSO:*R*(*k*), que nos explica que *k* unidades de recurso *R* son necesarias para la acción. El requerimiento de recurso es tanto un prerrequisito (la acción no puede ejecutarse si no tenemos dicho recurso disponible) y un efecto *temporal*, en el sentido de que la disponibilidad de un recurso *r* está reducida al valor *k*, para la duración de la acción. La Figura 12.3 muestra cómo extender el ejemplo del problema del ensamblaje del motor para incluir más recursos: un elevador para instalar moto-

RECURSO REUTILIZABLE

[1] En contraste, **recursos consumibles**, tales como tornillos para ensamblar el motor, pueden ser tratados como el resto de elementos del marco estándar; *véase* Ejercicio 12.2.

Iniciar (*Chasis*(C_1) ∧ *Chasis*(C_2)
∧ *Motor*(E_1, C_1, 30) ∧ *Motor*(E_2, C_2, 60)
∧ *Ruedas*(W_1, C_1, 30) ∧ *Ruedas*(W_2, C_2, 15)
∧ *Ensamblaje de carrocería* (*1*) ∧ *Ensamblaje de ruedas* (*1*) ∧ *Inspectores*(*2*))
Objetivo (*Montado*(C_1) ∧ *Montado*(C_2))

Acción (*AñadirMotor*(*e, c, m*)
PRECOND: *Motor*(*e, c, d*) ∧ *Chasis*(*c*) ∧ ¬*MotorDentro*(*c*),
EFECTO: *MotorDentro*(*c*) ∧ *Duración*(*d*)
RECURSO: *Ensamblaje de carrocería* (*1*))
Acción (*AñadirRuedas*(*w, c*),
PRECOND: *Ruedas*(*w, c, d*) ∧ *Chasis*(*c*),
EFECTO: *RuedasPuestas*(*c*) ∧ *Duración*(*d*),
RECURSO: *Ensamblaje de ruedas* (*1*))
Acción (*Inspeccionar*(*c*),
PRECOND: *MotorDentro*(*c*) ∧ *RuedasPuestas*(*c*),
EFECTO: *Montado*(*c*) ∧ *Duración*(10),
RECURSO: *Inspectores* (*1*))

Figura 12.3 Problema de programación de actividades para el ensamblaje de dos coches, con recursos. Los recursos disponibles son: una estación de ensamblaje de carrocería, una estación de ensamblaje de ruedas y dos inspectores. La notación RECURSO:*r* significa que el recurso *r* es usado durante la ejecución de una acción, pero vuelve a ser un recurso disponible cuando la acción se completa.

Figura 12.4 Solución al problema de programación de actividades para el ensamblaje de dos coches con recursos mostrado en la Figura 12.3. En el margen izquierdo se presenta una lista con tres recursos, y las acciones se muestran alineadas horizontalmente frente a los recursos que se consumen. Tenemos dos posibles programas de actividades dependiendo cuál de los dos ensamblajes utiliza la estación en primer lugar; se muestra la solución óptima que toma 115 minutos.

res, una estación adecuada para colocar las ruedas y dos inspectores que controlen los procesos. La Figura 12.4 muestra la solución más rápida, 115 minutos de tiempo completo. Esto es más que los 80 minutos requeridos para un programa sin limitación de recursos. Destaquemos que no es necesario, para ningún instante de tiempo, que ambos inspectores trabajen simultáneamente, de modo que podemos mover uno de ellos a posiciones más productivas.

AGREGACIÓN

La opción de representar los recursos como cantidades numéricas, tales como *Inspectores*(*2*), preferiblemente a entidades de tipo carácter, como *Inspector*(I_1) e *Inspector* (I_2), es un ejemplo de una técnica general que llamamos **agregación**. La idea central de la agregación es la agrupación de objetos individuales en diversas cantidades cuando los objetos son todos indistinguibles con respecto al propósito que tengamos entre manos. En nuestro problema del ensamblaje no existe problema en la elección de qué inspector inspecciona el coche, de modo que no necesitamos hacer distinciones. (La misma idea subyace en el problema de los misioneros y caníbales del Ejercicio 3.9.) La agregación es esencial para reducir la complejidad. Consideremos qué sucedería cuando una programación de tareas es propuesta con 10 acciones *Inspeccionar* simultáneas pero solamente nueve inspectores son disponibles. Con los inspectores representados como cantidades, un error es detectado inmediatamente y el algoritmo vuelve hacia atrás para intentar otra organización de tareas. Con los inspectores representados como individuo, el algoritmo regresa hacia atrás para intentar las 10 formas de asignar inspectores a las acciones *Inspeccionar*, en vano.

A pesar de sus ventajas, la restricción de recursos genera problemas más complicados porque introduce interacciones adicionales entre acciones. Mientras que la programación sin restricciones utilizando el método de camino-crítico es sencilla, encontrar un programa con limitación de tareas y con el mínimo tiempo posible es NP-duro. Esta complejidad es frecuentemente observada en la práctica y en la teoría. Un problema propuesto en 1963 (para encontrar el programa óptimo de un sistema que presentaba 10 máquinas y 10 trabajos de 100 acciones cada uno) se mantuvo sin resolver durante 23 años (Lawler *et al.,* 1993). Muchos enfoques fueron planteados, incluyendo técnicas de ramificación-y-poda, simulación, búsquedas, satisfacción de restricciones, y diversas técnicas de la Parte II. Una simple pero popular heurística es el algoritmo del **mínimo intervalo** de relajación. Este algoritmo, por cada iteración, considera las acciones no programadas que han tenido todos sus predecesores organizados y programa aquélla con menor intervalo de relajación para que se inicie lo antes posible. Actualiza los tiempos *ES* y *LS* para cada acción afectada y repite la operación. La heurística está basada sobre el mismo principio que la heurística de variable-más-restringida aplicada a problemas de satisfacción de restricciones. Frecuentemente tiene buenos resultados en la práctica, pero para nuestro problema del ensamblaje nos da una solución de 130 minutos, mayor que la solución de 115 minutos de la Figura 12.4.

MÍNIMO INTERVALO

El planteamiento que hemos tomado en esta sección es el de «primero planificar, después programar»: esto es, dividimos el problema completo en una fase de *planificación* en la que las acciones son seleccionadas y parcialmente ordenadas para alcanzar los objetivos del problema, y posteriormente una fase de *programación*, en la que la información temporal es añadida al plan para asegurarse que considera las limitaciones de recursos y los plazos. Este enfoque es común al que se toma en problemas logísticos y de manufacturado en el mundo real, donde la fase de planificación es normalmente ejecutada por expertos humanos. Cuando existen diferentes limitaciones de recursos, no obstante, podría suceder que algunos planes legítimos estén mucho mejor programados que otros. En ese caso, tiene sentido *integrar* la planificación y la organización tomando en consideración duración y solapamiento de acciones durante la construcción de un plan de orden parcial. Varios de los algoritmos de planificación expuestos en el Capítulo 11

pueden ser ampliados para trabajar con esta información. Por ejemplo, planificadores de orden-parcial pueden detectar que no se cumplen las restricciones de recursos del mismo modo que se detectan conflictos con enlaces causales. Las heurísticas pueden ser modificadas para estimar el tiempo total de un plan completo, en lugar de solamente estimar el coste total de las acciones. Esta es actualmente un área de investigación activa.

12.2 Redes de planificación jerárquica de tareas

DESCOMPOSICIÓN JERÁRQUICA

Una de las ideas dominantes en el tratamiento de problemas con alta complejidad es la **descomposición jerárquica**. Se diseña *software* complejo como jerarquías de subrutinas de clases-objeto; los ejércitos operan como unidades jerárquicas, los gobiernos y las corporaciones se estructuran como jerarquías de departamentos, servicios y oficinas. El beneficio central de las estructuras jerárquicas consiste en que, por cada nivel de la jerarquía, una tarea computacional, una misión militar, o función administrativa es reducida a un pequeño número de actividades en el siguiente nivel más bajo, de forma que el coste computacional para encontrar el modo correcto de planificar las actividades de un problema concreto es de poca importancia. En el lado contrario, tenemos métodos opuestos, llamados No-jerárquicos, y que reducen una tarea a un *amplio* número de acciones individuales; para problemas altamente escalables, es un planteamiento completamente impracticable. En el mejor de los casos (cuando soluciones de alto nivel se pueden satisfacer con implementaciones de bajo nivel) los métodos jerárquicos pueden presentar una dependencia lineal con el tiempo en lugar de las dependencias exponenciales de los algoritmos de planificación.

RED JERÁRQUICA DE TAREAS

Esta sección describe un método de planificación basado en **redes jerárquicas de tareas** o RJTs. El enfoque que tomamos combina ideas del ámbito de la planificación de orden-parcial (Sección 11.3) y el área conocida como «planificación RJT». En la planificación RJT, el plan inicial que describe el problema se ve como una descripción de alto nivel de lo que debe ser hecho (por ejemplo, construir una casa). Los planes son mejorados por la **descomposición de acciones**. Cada descomposición de una acción reduce una acción de alto nivel en un conjunto parcialmente ordenado de acciones de menor nivel. La descomposición de acciones, por tanto, incorpora conocimiento acerca de cómo implementar acciones. Por ejemplo, construir una casa podría ser reducido a obtener un permiso, disponer de los servicios de un contratista, ejecutar las obras y pagar al contratista. (En la Figura 12.5 mostramos esta descomposición.) El proceso continúa hasta que solamente queden en el plan **acciones primitivas**. Típicamente, las acciones primitivas serán acciones que el agente podrá ejecutar automáticamente. Para un contratista genérico, «diseñar jardines» puede ser primitivo porque simplemente se limita a llamar al jardinero. Para el jardinero, acciones tales como «plantar petunias» pueden considerarse primitivas.

DESCOMPOSICIÓN DE ACCIONES

ACCIÓN PRIMITIVA

En planificación RJT «pura», los planes son generados *solamente* para descomposición de acciones sucesivas. La RJT ve la planificación como un proceso más *concreto* de confeccionar la descripción de actividades, en lugar de (como en el caso del espacio de estados y de la planificación de orden parcial) un proceso de *construcción* de descripciones

de actividades, iniciado con la actividad vacía. Esto nos lleva a que cada descripción de acciones STRIPS pueda ser reducida a una descomposición de acciones (*véase* Ejercicio 12.6), y que la planificación de orden-parcial pueda ser vista como un caso especial de planificación RJT pura. Para ciertas tareas, sin embargo (especialmente las secuencias de objetivos «noveles») el punto de vista RJT puro es poco natural, y preferimos tomar un enfoque *híbrido* en el que la descomposición de acciones sea usada como refinamientos de planes en planificación de orden parcial, además de operaciones estándar como la de establecer condiciones abiertas y resolver conflictos añadiendo restricciones ordenadas (considerando la planificación RJT como una extensión de la planificación de orden parcial, se tiene la ventaja adicional de que podemos usar las mismas convenciones sobre notación en lugar de introducir un nuevo conjunto de símbolos). Comencemos por describir la descomposición de acciones en mayor detalle. Así explicaremos cómo algoritmos de planificación de orden-parcial deben ser modificados para trabajar con descomposiciones, y finalmente discutiremos sobre problemas de completitud, complejidad y utilidad.

Representación de descomposición de acciones

BIBLIOTECA DE PLANES

Las descripciones generales de los métodos de descomposición de acciones se almacenan en una **biblioteca de planes**, desde donde se recuperan, extrayendo e instanciando elementos que cumplan las necesidades del plan que está siendo construido. Cada método es una expresión de la forma *Descomponer*(*a, d*). Esto nos informa de que una acción *a* puede ser descompuesta dentro del plan *d*, lo que se representa como un plan de orden parcial, tal como describimos en la Sección 11.3.

Construir una casa es un ejemplo concreto y sencillo que usaremos para ilustrar este concepto de descomposición de acciones. La Figura 12.5 representa una posible descomposición de la acción *ConstruirCasa* en cuatro acciones de menor nivel. La Figura 12.6 muestra alguna de las descripciones de acciones en el dominio y la descomposición de *ConstruirCasa* como si apareciese en la biblioteca de planes. Podría existir más de una forma de descomposición posible siguiendo la biblioteca de planes.

PRECONDICIONES EXTERNAS

EFECTOS EXTERNOS

EFECTOS PRIMARIOS

EFECTOS SECUNDARIOS

La acción *Inicio* de la descomposición proporciona todas aquellas precondiciones de las acciones del plan que no son suministradas por otras acciones. Llamaremos a éstas **precondiciones externas**. En nuestro ejemplo, las precondiciones externas de la descomposición son *Tierra* y *Dinero*. De forma equivalente, los **efectos externos**, que son las precondiciones de *Finalizar*, son todos aquellos efectos de las acciones del plan que no son negados por otras acciones. En nuestro ejemplo, los efectos externos de *ConstruirCasa* son *Casa*, y ¬*Dinero*. Algunos planificadores RJT también distinguen entre **efectos primarios**, tales como *Casa,* y **efectos secundarios** tales como ¬*Dinero*. Solamente los efectos primarios pueden ser usados para alcanzar objetivos, mientras que ambas clases de efectos pueden causar conflictos con otras acciones; esto puede reducir considerablemente el espacio de estados[2].

[2] También podría prevenir el descubrimiento de planes inesperados. Por ejemplo, una persona que haga frente a una posible bancarrota puede eliminar todos sus activos líquidos (es decir, alcanzar el estado ¬*Dinero*) por construir o comprar una casa. Este plan es útil porque la actual ley excluye la confiscación de la primera vivienda por los acreedores.

Figura 12.5 Posible descomposición para la acción *ContruirCasa*.

Acción (*ComprarTierra,* PRECOND: *Dinero,* EFECTO: *Tierra* $\wedge$ $\neg$*Dinero*)
Acción (*ConseguirPréstamo,* PRECOND: *BuenCrédito,* EFECTO: *Dinero* $\wedge$ *Hipoteca*)
Acción (*ConstruirCasa,* PRECOND: *Tierra,* EFECTO: *Casa*)

Acción (*ObtenerPermiso,* PRECOND: *Tierra,* EFECTO: *Permiso*)
Acción (*ContratarConstructor,* EFECTO: *Contrato*)
Acción (*Construcción,* PRECOND: *Permiso* $\wedge$ *Contrato,*
EFECTO: *CasaConstruida* $\wedge$ $\neg$*Permiso*)
Acción (*PagarConstructor,* PRECOND: *Dinero* $\wedge$ *CasaConstruida,*
EFECTO: $\neg$ *Dinero* $\wedge$ *Casa* $\wedge$ $\neg$*Contrato*)

Descomponer(*ConstruirCasa,*
Plan(ETAPAS:{S_1: *ObtenerPermiso,* S_2: *ContratarConstructor,*
S_3: *Construcción,* S_4: *PagarConstructor*}
ORDEN: {$Inicio \prec S_1 \prec S_3 \prec S_4 \prec Finalizar,\ Inicio \prec S_2 \prec S_3$},
ENLACES: {$Inicio \xrightarrow{Tierra} S_1,\ Inicio \xrightarrow{Dinero} S_4,$
$S_1 \xrightarrow{Permiso} S_3,\ S_2 \xrightarrow{Contrato} S_3,\ S_3 \xrightarrow{CasaConstruida} S_4,$
$S_4 \xrightarrow{Casa} Finalizar,\ S_4 \xrightarrow{\neg Dinero} Finalizar$}))

Figura 12.6 Descripción de acciones para el problema de la construcción de una casa y descomposición detallada para la acción *ConstruirCasa*. Las descripciones adoptan una visión simplificada sobre dinero y una visión optimista sobre los constructores.

Una descomposición debería ser una *correcta* implementación de la acción. Un plan *d* implementa correctamente una acción *a,* si *d* es un completo y consistente plan de orden parcial para el problema de alcanzar los efectos de *a,* dadas las precondiciones de *a*. Obviamente, una descomposición será correcta si es el resultado de ejecutar un plan de orden parcial adecuado.

Una biblioteca de planes podría contener varias descomposiciones para cualquier acción de alto nivel; por ejemplo, podría existir otra descomposición para *ConstruirCasa* que describa un proceso por el cual el agente construya una casa de rocas y barro con sus manos desnudas. Cada descomposición debe ser un plan correcto, pero podría tener

precondiciones adicionales y efectos más allá que los que estaban establecidos en la descripción de acciones de alto nivel. Por ejemplo, la descomposición para *ContruirCasa* de la Figura 12.5 requiere *Dinero* y *Tierra* y tiene el efecto ¬*Dinero*. La opción de construir personalmente la casa, por otro lado, no requiere dinero, pero necesita un suministro de *Rocas* y *Barro*, lo cual puede ser contraproducente.

Dada una acción de alto nivel, como la de *ContruirCasa*, pueden existir varias posibles descomposiciones, y es inevitable que la descripción de la acción por STRIPS posea algunas de las precondiciones y efectos de tales descomposiciones. Las precondiciones de una acción de alto nivel, deben ser la *intersección* de las precondiciones externas de sus descomposiciones, y los efectos deben ser la intersección de los efectos externos de sus descomposiciones. Dicho de otra forma, las precondiciones y los efectos de alto nivel tienen asegurado ser un subconjunto de las precondiciones y efectos de cada implementación primitiva.

EFECTOS INTERNOS

Otros dos modos de explotar información deben ser mencionados. En primer lugar, la descripción de alto nivel ignora completamente todos los **efectos internos** de la descomposición. Por ejemplo, nuestra descomposición de *ConstruirCasa* tiene efectos internos provisionales como *Permitir* y *Contratar*[3]. En segundo lugar, la descripción de alto nivel no especifica los intervalos «dentro» de la actividad durante los que las precondiciones de alto nivel son efectos que deben cumplirse. Por ejemplo, la precondición *Tierra* necesita ser verdad (en nuestro modelo aproximado) sólo hasta que *ObtenerPermiso* sea ejecutado, y *Casa* será verdad sólo después de que sea ejecutado *PagarConstructor*.

La información que se trata en este enfoque es esencial si la planificación jerárquica pretende reducir la complejidad; necesitamos ser capaces de razonar acerca de acciones de alto nivel sin preocuparnos acerca de miles de detalles de las implementaciones. Existe, sin embargo, un precio que hay que pagar. Por ejemplo, pueden existir conflictos entre condiciones internas de una acción de alto nivel y las de otra, pero no hay modo de detectar esto desde las descripciones de alto nivel. Este asunto tiene implicaciones significativas para algoritmos de planificación RJT. En resumen, mientras que las acciones primitivas puedan ser tratadas como eventos puntuales por el algoritmo de planificación, las acciones de alto nivel tendrán entidad temporal dentro de la cual puede suceder todo de tipo de cosas.

Modificación de planificadores para su descomposición

A continuación mostraremos cómo modificar POP para incorporar planificación RJT. Lograremos esto mediante la modificación de la función de sucesores POP (Apartado 11.3) para permitir que métodos de descomposición sean aplicados al plan parcial actual P. Los nuevos planes sucesores están formados primeramente mediante la selección de algunas acciones no primitivas a' en P y posteriormente, para cada método *Descomponer(a, d)* de la librería de planes de modo que a y a' se unifican mediante la substitución θ, reemplazando a' por $d' = \text{SUBST}(\theta, d)$.

[3] *Construcción* niega el *Permiso*, aparte de que el mismo permiso puede ser usado para construir muchas casas. Desafortunadamente, *Construcción* no pone fin al *Contrato* porque tenemos que *PagarConstructor* primero.

La Figura 12.7 nos muestra un ejemplo. Inicialmente, tenemos un plan P para adquirir una casa. La acción de nivel alto, $a' = ConstruirCasa$, se selecciona para descomponerse. La descomposición d es seleccionada de la Figura 12.5, y *ConstruirCasa* es reemplazada por su descomposición. En una etapa posterior, *AdquirirPréstamo* se introduce para resolver la nueva condición abierta, *Dinero*, que es creada a partir de la descomposición. Reemplazar una acción por su descomposición es un pequeño ejercicio de transplante quirúrgico: debemos eliminar el nuevo subplan de su envoltorio (las etapas *Inicio* y *Final*), insertarlo, y «engancharlo» todo apropiadamente. Podemos emplear diferentes modos de hacer esto. Para ser más precisos, tenemos para cada posible descomposición d',

Figura 12.7 Descomposición de una acción de alto-nivel dentro de un plan existente. La acción *ConstruirCasa* es reemplazada por su descomposición indicada en la Figura 12.5. La precondición externa *Tierra* es proporcionada por la conexión causal desde *ComprarTierra*. La precondición externa *Dinero* permanece abierta después de la fase de descomposición, de modo que añadimos una nueva acción, *AdquirirPréstamo*.

1. En primer lugar, la acción a' se elimina de P. Entonces, para cada etapa s en la descomposición d', necesitamos elegir una acción que ocupe el rol de s y añadirla al plan. Puede consistir en una nueva instanciación de s o en una fase existente s' de P que se unifique con s. Por ejemplo, la descomposición de la acción *HacerVino* podría sugerir la acción *ComprarTierra*; posiblemente, podemos utilizar la misma acción *ComprarTierra* que ya teníamos en el plan. Llamamos a este proceso **distribución de subtareas**.

DISTRIBUCIÓN DE SUBTAREAS

En la Figura 12.7 no hay posibilidades de distribución, de modo que son creadas nuevas instancias de acciones. Una vez que las acciones han sido elegidas, todas las restricciones internas de d' son transcritas (por ejemplo, que *AdquirirPréstamo* sea ordenado antes de *Construcción* y que exista una relación causal entre estas dos etapas proporcionando *Permiso* como precondición de *Construcción*). Esto completa la tarea de reemplazar a' con la instanciación de $d\theta$.

2. La siguiente fase es conectar las restricciones de orden de a' en el plan original con las fases en d'. Primeramente, consideramos una restricción de orden en P de la forma $B \prec a'$. ¿Cómo debe ser ordenada B con respecto a las fases en d'? La solución más obvia es que B comparezca en cada etapa de d', y esto puede ser alcanzado reemplazando cada restricción de la forma $Inicio \prec s$ en d', con una restricción $B \prec s$. Por otro lado, este enfoque puede ser demasiado rígido. Por ejemplo, *ComprarTierra* tiene que prevalecer a *ConstruirCasa*, pero no hay necesidad para que *ComprarTierra* venga antes de *ContratarConstructor* en el plan expandido. Imponer una orden demasiado estricta puede impedir la posibilidad de encontrar algunas soluciones. Por tanto, la mejor solución consiste en registrar, para cada restricción de orden, cuál es la *razón* de la restricción; por tanto, cuando una acción de alto nivel sea expandida, las nuevas restricciones de orden pueden ser tan flexibles como sea posible, consistentes con la restricción original. Exactamente ocurre lo mismo cuando reemplazamos restricciones de la forma $a' \prec C$.
3. La última fase es la conexión de los enlaces causales. Si $B \xrightarrow{p} a'$ era un enlace causal en el plan original, se reemplaza por un conjunto de enlaces causales desde B hacia todas las fases en d' con precondiciones p que fueron suministradas por la etapa *Iniciar* en la descomposición d (esto es, para todas las fases en d' para las cuales p sea una precondición externa). En el ejemplo, la conexión causal, $ComprarTierra \xrightarrow{Tierra} ConstruirCasa$ es reemplazada por $ComprarTierra \xrightarrow{Tierra} Permiso$ (la precondición *Dinero* para *PagarConstructor* en la descomposición se transforma en una condición abierta, porque ninguna acción en el plan original suministra *Dinero* para *ConstruirCasa*). De manera similar, para cada conexión causal $a' \xrightarrow{p} C$ en el plan, se reemplaza con un conjunto de enlaces causales a C desde cualquier fase en d' que suministra p a la fase *Finalizar* en la descomposición d (es decir, desde la etapa en d' que tiene a p como un efecto *externo*). En el ejemplo, se reemplaza la relación $ConstruirCasa \xrightarrow{Casa} Finalizar$ por el enlace $PagarConstructor \xrightarrow{Casa} Finalizar$.

Esto completa todos los elementos requeridos para generar descomposiciones en el contexto del planificador POP[4].

Modificaciones adicionales al algoritmo POP son requeridas por el hecho de que las acciones de alto nivel ocultan información acerca de sus definitivas implementaciones de primitivas. En particular, el algoritmo original POP retrocede con errores si el plan actual contiene un conflicto irresoluble, esto es, si la acción entra en conflicto con un enlace causal pero no puede ser ordenado ni antes ni después del mismo (la Figura 11.9 muestra un ejemplo). Con acciones de alto nivel, por el otro lado, conflictos aparentemente irresolubles pueden a veces ser resueltos mediante *descomposición* de acciones conflictivas e intercalando sus etapas. Un ejemplo se muestra en la Figura 12.8. De este modo, puede darse el caso de que un plan completo y consistente pueda ser obtenido por descomposición *incluso aun cuando no exista plan completo y consistente de alto nivel*.

[4] Existen algunas modificaciones menores requeridas para tratar resolución de conflictos con acciones de alto nivel; los lectores interesados pueden consultar los trabajos citados al final del capítulo.

Figura 12.8 El problema del *Regalo de los Reyes Magos*, tomado de la historia de O. Henry muestra un plan abstracto e inconsistente que sin embargo puede descomponerse en soluciones consistentes. La parte (a) muestra el problema: Una pobre pareja tiene sólo dos posesiones preciosas: él un reloj de oro y ella una bella melena rubia. Cada uno planea comprar un regalo que haga al otro feliz. Él decide cambiar su reloj por un peine de plata para su pelo, y ella decide vender su pelo para comprar una correa de oro para su reloj. En (b) el plan parcial es inconsistente, porque no hay manera de ordenar «DarPeine» y «DarCorrea» como fases abstractas sin un conflicto. (Se asume que la acción «DarPeine» tiene la precondición *Pelo*, porque si la mujer no tuviese el pelo largo, la acción no tendría la intención de hacerla feliz, y de modo similar la acción «DarCadena».) En (c) se descompone la fase «DarCadena» con un método del tipo «plan a plazos». En la primera fase de la descomposición, el marido adquiere el peine y se lo da a su mujer, mientras que acuerda entregar el reloj como pago un día después. En la segunda fase, el reloj es entregado y la obligación adquirida se cumple. Un método similar descompone la etapa «DarCorrea». Tan pronto como ambas fases sean ordenadas antes de los momentos de entrega, esta descomposición resuelve el problema. (Nótese que esto descansa sobre la idea de definición del problema, de modo que la felicidad del uso de la correa con el reloj, o del peine con el pelo, persiste aún incluso después de que las posesiones sean entregadas.)

Esta posibilidad significa que un planificador completo RJT tiene que renunciar a muchas oportunidades de poda que se encuentran disponibles para un planificador POP estándar. Alternativamente, podemos utilizar procesos de poda indiscriminadamente y esperar que ninguna solución se pierda.

Discusión

Comencemos con las malas noticias: la planificación RJT pura (donde el único modo de refinamiento de planes permitido es la descomposición) es indecidible, *incluso aunque el espacio de estados subyacentes sea finito*. Esto podría parecer muy deprimente,

dado que el fin de la planificación RJT es lograr eficiencia. La dificultad crece porque la descomposición de las acciones puede ser **recursiva** (por ejemplo, dar un paseo, puede ser implementado como dar un paso y comenzar a dar un paseo) de modo que los planes RJT pueden ser arbitrariamente largos. En particular, la solución RJT más corta puede ser arbitrariamente larga, de modo que no hay manera de terminar la búsqueda para un tiempo fijado. Existen, sin embargo, al menos tres maneras de enfocar la solución:

1. Podemos descartar la recursión que en muy pocos dominios se requiere. En ese caso, todos los planes RJT son de duración finita y pueden ser enumerados.
2. Podemos limitar la longitud de soluciones que nos preocupen. Como el espacio de estados es finito, un plan que tenga más etapas que estados en el espacio de estados *debe* incluir un bucle que visite el mismo estado dos veces. Podremos perder poco si descartamos soluciones RJT de este tipo, y podremos controlar la búsqueda.
3. Podemos adoptar un enfoque híbrido que combine planificación POP y RJT. La planificación de orden parcial es por sí misma suficiente para decidir si un plan existe, de modo que el problema híbrido es claramente decidible.

Necesitamos tener un cuidado mayor con esta tercera opción. POP puede coordinar acciones primitivas de maneras diferentes, de modo que podríamos encontrarnos con soluciones que son muy duras de entender y que no tienen la organización jerárquica de los planes RJT. Un compromiso apropiado consiste en controlar la búsqueda híbrida de modo que la descomposición de acciones sea preferible a la adición de nuevas acciones. Una forma de hacer esto es usar una función de coste que nos proporcione un descuento para las acciones introducidas mediante descomposición; cuanto mayor sea dicho descuento, mayor será la similitud de la búsqueda con planificación RJT y la solución será más jerárquica. Los planes jerárquicos son normalmente mucho más sencillos de ejecutar en sistemas realistas, y son más sencillos de reparar cuando las cosas van mal.

Otra característica importante de los planes RJT es la posibilidad de distribuir subtareas. Recordemos que distribuir subtareas significa el uso de la misma acción para implementar dos etapas diferentes en la descomposición de un plan. Si prohibimos la distribución de subtareas, entonces cada instanciación de una descomposición d' puede ser hecha de un solo modo, preferiblemente que de varios por esta razón en gran medida se poda el espacio de búsqueda. Normalmente estos procesos de poda economizan tiempo y en el peor de los casos nos llevan a soluciones ligeramente más largas que la óptima. Por ejemplo, consideremos el objetivo «disfrutar de una luna de miel y formar una familia». La librería de planes podría comenzar con «casarse y viajar a Hawai», para el primer subobjetivo, y «casarse y tener dos hijos» para el segundo objetivo. Sin distribución de tareas, el plan incluirá dos acciones de matrimonio diferentes, lo que frecuentemente es considerado como altamente indeseable.

Un ejemplo interesante de los costes y beneficios de la distribución de subtareas ocurre en la optimización de compiladores. Consideremos el problema de compilar la expresión $\tan(x) - \sin(x)$. Muchos compiladores logran esto mediante la combinación de dos subrutinas separadas de un modo trivial: todas las etapas de $\tan(x)$ se presentan

antes que las fases de sin (x). Pero considerando la siguiente aproximación de Taylor a las funciones sin (x) y tan (x):

$$\tan x \approx x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315}; \qquad \sin x \approx x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5.040}$$

Un planificador RJT con distribución de subtareas podría generar una solución más eficiente, porque podría seleccionar para la implementación de muchas etapas de sin (x) algunas etapas existentes de tan (x). Muchos compiladores no permiten esta clase de distribución interprocedural porque podría necesitar mucho tiempo para considerar todos los posibles planes compartidos. En lugar de eso, la mayoría de los compiladores generan cada subplan independientemente, y después quizá modifican el resultado con un optimizador más concreto.

Dadas todas las complicaciones adicionales causadas por la introducción de la descomposición de acciones, ¿por qué creemos que la planificación RJT es eficiente? Las fuentes actuales de complejidad son duras de analizar en la práctica, de modo que consideraremos un caso ideal. Supongamos, por ejemplo, que queremos construir un plan con n acciones. Para un planificador no jerárquico con b acciones posibles en cada estado, el coste es $O(b^n)$. Para un planificador RJT, supongamos una estructura de descomposición muy regular: cada acción no primitiva tiene d posibles descomposiciones, cada una con k acciones en el nivel inferior siguiente. Queremos saber cuántos árboles de diferentes descomposiciones existen con esa estructura. Si existen n acciones en el nivel más primitivo, entonces el orden de los niveles bajo la raiz es $\log_k n$, de forma que el número de nodos de descomposición interna es $1 + k + k^2 + \ldots + k^{\log_k n - 1} = (n - 1)/(k - 1)$. Cada nodo interno tiene d posibles descomposiciones, de forma que hay $d^{(n-1)/(k-1)}$ posibles descomposiciones de árboles regulares que pueden ser construidos. Examinando esta fórmula, vemos que mantener a d pequeño y a k alto puede traducirse en grandes ahorros: esencialmente estamos tomando la raíz kth del coste no-jerárquico, si b y d son comparables. Del otro lado, la construcción de una librería de planes que tenga un pequeño número de grandes descomposiciones, pero aún con todo nos permita resolver problemas, no es siempre posible. Otra forma verlo: grandes macros que son útiles a lo largo de un amplio espectro de problemas son extremadamente valiosos.

Otra razón, y quizá mejor, para creer que la planificación RJT es eficiente es que funciona en la práctica. Casi todos los planificadores para aplicaciones de escala larga son planificadores RJT, porque los planificadores RJT permiten a los expertos humanos proporcionar el conocimiento crucial acerca de cómo tratar tareas complejas de forma que planes grandes puedan ser construidos con poco esfuerzo computacional. Por ejemplo, O-PLAN (Bell y Tate, 1985), que combina planificación RJT con programación de tareas, ha sido utilizado para desarrollar planes de producción para Hitachi. Un problema típico relaciona una línea de productos de 350 productos diferentes, 35 máquinas de ensamblaje, y del orden de 2.000 operaciones diferentes. El planificador genera un programa de 30 días de duración con ocho horas al día, implicando a millones de etapas.

La clave de la planificación RJT, por tanto, está en la construcción de una librería de planes conteniendo métodos conocidos para la implementación de acciones complejas de alto nivel. Un método para construir la librería es aprender los métodos extraídos de la experiencia de resolución de problemas previos. Después de escrutar la experien-

cia en la construcción de planes, el agente puede guardar el plan en la librería como un método para implementar la acción de alto nivel definida por la tarea. De este modo, el agente puede ser cada vez más y más competente a lo largo del tiempo cuando nuevos métodos son diseñados sobre métodos antiguos. Un aspecto importante de este proceso de aprendizaje es la habilidad para *generalizar* los métodos que son construidos, eliminando detalles que son específicos de la instancia del problema (por ejemplo, el nombre del constructor o la dirección del terreno edificable) y manteniendo simplemente los elementos claves del plan. Los métodos para alcanzar esta clase de generalización son descritos en el Capítulo 19. Sería inconcebible que los humanos pudiesen ser tan competentes como lo son sin algunos de estos mecanismos.

12.3 Planificación y acción en dominios no deterministas

Hasta este momento hemos considerado exclusivamente dominios de **planificación clásica** que sean completamente observables, estáticos y deterministas. Además de esto, hemos asumido que las descripciones de las acciones son correctas y completas. En estas circunstancias, un agente puede planificar primero y después ejecutar el plan «con los ojos cerrados». En un entorno con incertidumbre, al contrario, un agente debe usar sus percepciones para descubrir qué está sucediendo mientras el plan está siendo ejecutado y posiblemente modifique o reemplace el plan si algo inesperado sucede.

Los agentes tienen que tratar tanto con información *incompleta* como *incorrecta*. La incompletitud surge porque el mundo es parcialmente observable, no determinista, o ambas cosas. Por ejemplo, la puerta para acceder a la oficina puede o no estar cerrada; una de mis llaves podría o no abrir la puerta en el caso de que estuviese cerrada; y yo podría ser consciente o no de esta clase de incompletitud en mi conocimiento. De este modo, mi modelo del mundo es débil pero correcto. Por otro lado, la incorrección surge porque el mundo no se comporta necesariamente como mi modelo predice; por ejemplo, puedo *creer* que mi llave abrirá la puerta de la oficina, pero podría estar equivocado si las cerraduras han sido cambiadas. Sin la habilidad para tratar con información incorrecta, un agente puede llegar a ser más torpe que un escarabajo estercolero (Apartado 2.2), que intenta tapar su hormiguero con una bola de estiércol incluso aunque durante el trayecto le hayamos apartado la bola fuera de su alcance.

INDETERMINACIÓN LIMITADA

La posibilidad de tener conocimiento completo y correcto depende de *cuánta* indeterminación hay en el mundo. Con **indeterminación limitada**, las acciones pueden tener efectos impredecibles, pero los posibles efectos pueden ser ordenados en los axiomas de descripción de acciones. Por ejemplo, cuando lanzamos una moneda, es razonable decir que el resultado será *Cara* o *Cruz*. Un agente puede hacer frente a la indeterminación limitada diseñando planes que trabajen en todas las posibles circunstancias. Con la **indeterminación ilimitada**, por otro lado, el conjunto de posibles precondiciones o efectos o es desconocido o es demasiado extenso para ser enumerado completamente. Este podría ser el caso en entornos muy complejos o dominios dinámicos tales como con-

INDETERMINACIÓN ILIMITADA

ducir, planificación económica y estrategia militar. Un agente puede hacer frente a indeterminación ilimitada solamente si está preparado para revisar sus planes y/o su conocimiento base. La indeterminación ilimitada está estrechamente relacionada con el **problema de cualificación** discutido en el Capítulo 10 (la imposibilidad de hacer una lista con todas las precondiciones requeridas para acciones en el mundo real que tengan los efectos buscados).

Existen cuatro métodos de planificación para trabajar con indeterminación. Los dos primeros son adecuados para la indeterminación limitada y los dos segundos para la indeterminación ilimitada:

PLANIFICACIÓN SIN SENSORES

— **Planificación sin sensores**: es conocida también como **planificación conformista**, y es un método que construye planes estándares y secuenciales, diseñados para ser ejecutados sin necesidad de percepción. Los algoritmos de planificación sin sensores deben asegurar que el plan alcanza el objetivo en *todas las posibles circunstancias*, independientemente de si es correcto el estado inicial y los resultados actuales de las acciones. La planificación sin sensores descansa sobre la **coerción** (la idea de que el mundo puede estar restringido a un estado dado incluso cuando el agente tiene sólo información parcial acerca del estado actual). La coerción no siempre es posible, de modo que la planificación sin sensores es frecuentemente inaplicable. La resolución de problemas sin sensores, que engloban las búsquedas en espacios de estados de creencias, fueron descritos en el Capítulo 3.

PLANIFICACIÓN CONDICIONAL

ACCIONES SENSORIALES

— **Planificación condicional**: también conocida como **planificación contingente**; este enfoque trata con indeterminación limitada mediante la construcción de un plan condicional con diferentes ramas para las diferentes contingencias que puedan surgir. Al igual que en la planificación clásica, el agente hace planes primero, y después ejecuta el plan que ha diseñado. El agente encuentra cuál es la parte del plan que debe ejecutar, incluyendo **acciones sensoriales** que evalúen las condiciones apropiadas. En el problema del transporte aéreo, por ejemplo, podríamos tener planes que ordenasen «Evaluar si el aeropuerto SFO está operativo. Si es así, volar allí; si no es así, volar a Oakland». La planificación condicional se explica en la Sección 12.4.

VIGILANCIA DE EJECUCIÓN Y PLANIFICACIÓN

— **Vigilancia de ejecución y replanificación**: en este enfoque, el agente puede usar cualquiera de las técnicas de planificación previamente presentadas (clásica, sin sensores o condicional) para construir un plan, pero además emplea un procedimiento de **vigilancia de ejecución** para juzgar si el plan tiene capacidad de cumplirse dado el estado actual, o necesita ser revisado. La **replanificación** ocurre cuando algo va mal. De esta forma, el agente tiene que afrontar indeterminación ilimitada. Por ejemplo, incluso si un agente con capacidad para replanificar no considera en un momento la posibilidad de que el SFO esté cerrado, ahora es capaz de reconocer la situación nueva cuando ocurre y acudir de nuevo al planificador para encontrar un nuevo modo de alcanzar el objetivo. Los agentes con capacidad de replanificación serán estudiados en la Sección 12.5.

PLANIFICACIÓN CONTINUA

— **Planificación continua**: todos los planificadores que hemos visto hasta ahora están diseñados para alcanzar un objetivo y después detenerse. Un planificador continuo está diseñado para persistir a lo largo del tiempo. Puede tratar con cir-

cunstancias inesperadas en el entorno, incluso si ocurre cuando el agente está en el punto medio del desarrollo de un plan. También pueden abordar el abandono de objetivos y la creación de objetivos adicionales mediante **formulación de objetivos**. Estudiaremos los mecanismos de planificación continua en la Sección 12.6.

Consideremos un ejemplo para clarificar las diferencias entre los distintos tipos de agentes. El problema es el siguiente: dado un estado inicial con una silla, una mesa y algunas latas de pintura, de colores desconocidos, alcanzar el estado donde la silla y la mesa tengan el mismo color.

Un agente de **planificación clásico** no podría tratar este problema, porque el estado inicial no está completamente especificado (no conocemos de qué color son los muebles).

Un agente de **planificación sin sensores** debería encontrar un plan que actuase sin la necesidad de requerir sensores durante la ejecución del plan. La solución consiste en abrir todas las latas de pintura y pintar tanto la silla como la mesa, de este modo **forzamos** que tengan el mismo color (incluso sin que el agente conozca qué color es). La idea de coerción es la más apropiada cuando las proposiciones son imposibles de percibir. Por ejemplo, los médicos frecuentemente prescriben antibióticos de amplio espectro preferiblemente al uso de un plan condicional que consista en realizar una muestra de sangre, esperar los resultados, y después prescribir un antibiótico más específico. Se hace esto porque los retrasos y los costes de las pruebas de sangre son normalmente elevados.

Un agente de **planificación condicional** puede generar mejores planes: primeramente, es sensible al color de la mesa y la silla; si los colores son iguales, el plan está terminado. Si no coinciden los colores, es sensible a las etiquetas que figuran en las latas de pintura; si una de las latas es del mismo color que uno de los elementos, mesa o silla, entonces pintamos de dicho color la otra pieza. En otro caso, ejecutará la orden de pintar ambos elementos del mismo color.

Un agente de **replanificación** puede generar el mismo plan que el planificador condicional, o podría generar versiones más cortas, en principio, y acudir al resto en tiempo de ejecución si fuese necesario. Podría tratar con la incorrección de su propia descripción de acciones. Por ejemplo, supongamos que creemos que la acción *Pintar* (*objeto, color*) tiene de manera determinista el efecto *Color*(*objeto, color*). Un planificador condicional podría sencillamente asumir que el efecto ha ocurrido una vez que la acción ha sido ejecutada, pero un agente de replanificación podría evaluar el efecto, y si no fuese correcto (quizá porque el agente no haya sido cuidadoso y haya olvidado un lugar), podría entonces replanificar y pintar ese lugar olvidado. Regresaremos a este ejemplo en el Apartado 12.5.

Un agente de **planificación continua**, además de tratar con sucesos inesperados, puede revisar sus propios planes adecuadamente si, digamos, añadimos el objetivo de cenar sobre la mesa, de modo que el plan para pintarla debe ser pospuesto.

En el mundo real, los agente usan una combinación de estos diferentes enfoques. Los constructores de coches venden ruedas de repuesto y *airbags*, que son una manifestación física de algunas ramas de planes condicionales diseñados para tratar con pinchazos o accidentes; por otro lado, gran parte de los conductores nunca consideran estas posibilidades, de modo que responden ante pinchazos y accidentes como agentes de re-

planificación. En general, los agentes crean planes condicionales sólo frente a aquellas contingencias que tengan importantes consecuencias. De esta forma, un conductor de coches contemplando la posibilidad de un viaje a lo largo del desierto del Sahara debe considerar explícitamente la posibilidad de perturbaciones, mientras que un viaje al supermercado requiere menos planificación previa.

Los agentes que describimos es este capítulo están diseñados para tratar con indeterminación, pero no son capaces de plantear intercambios entre la probabilidad de éxito y el coste de la construcción de un plan. En el Capítulo 16 presentaremos herramientas adicionales para tratar con estas situaciones.

12.4 Planificación condicional

La planificación condicional es un modo de tratar con la incertidumbre evaluando qué está realmente sucediendo en el entorno a predeterminados instantes del plan. La planificación condicional es la más simple para explicar los entornos completamente observables, de modo que comenzaremos con este caso. El caso parcialmente observable es más complicado, pero más interesante.

Planificación condicional en entornos completamente observables

La completa observabilidad en un entorno significa que el agente siempre conoce el estado actual. Si el entorno es no-determinista, el agente no será capaz de predecir el *resultado* de sus acciones. Un agente de planificación condicional ataca el no-determinismo mediante la construcción dentro de un plan (en tiempo de planificación) de etapas condicionales que evaluarán el estado del entorno (a tiempo de ejecución) para decidir qué hacer en el instante siguiente. El problema, por tanto, es cómo construir estos planes condicionales.

Utilizaremos como entorno de trabajo en nuestros ejemplos el **mundo aspiradora,** cuyo espacio de estados, para el caso determinista, se presenta en el Apartado 3.2. Recordemos que las acciones disponibles son *Izquierda, Derecha,* y *Aspirar.* Necesitamos algunas proposiciones para definir los estados: sugerimos que *AtI*(*AtD*) sea correcta si el agente está en el estado izquierdo (derecho)[5], y que *CleanI*(*CleanD*) sea correcta si el estado izquierdo (derecho) está limpio. Lo primero que necesitamos es incrementar la expresividad del lenguaje STRIPS para tener en cuenta el no-determinismo. Para hacer esto, utilizamos acciones que tengan **efectos disyuntivos**, esto es, que la acción puede tener dos o más resultados diferentes en cualquier momento en que sea ejecutado. Por ejemplo, supongamos que moverse hacia *Izquierda* algunas veces fracase. Por tanto, la descripción normal de la acción

EFECTOS DISYUNTIVOS

Acción(*Izquierda,* PRECOND:*AtD,* EFECTO:*AtI*)

[5] Obviamente, *AtD* es correcto si y sólo si ¬*AtI* es verdadero, y viceversa. Utilizamos dos proposiciones para mejorar la legibilidad.

debe ser modificada para incluir un efecto disyuntivo:

$$\textit{Acción}\ (\textit{Izquierda},\ \text{PRECOND}{:}\textit{AtD},\ \text{EFECTO}{:}\textit{AtI} \vee \textit{AtD}) \qquad (12.1)$$

EFECTOS CONDICIONALES

También encontramos útil permitir acciones que tengan **efectos condicionales**, donde el efecto de la acción depende del estado en el que sea ejecutado. Los efectos condicionales aparecen en el slot EFECTO de una acción, y tienen la sintaxis «**cuando** <*condición*>: <*efecto*>». Por ejemplo, para modelar la acción *Aspirar,* expresamos

$$\textit{Acción}\ (\textit{Aspirar},\ \text{PRECOND}{:},\ \text{EFECTO}{:}(\textbf{cuando}\ \textit{AtI: LimpiarI}) \wedge (\textbf{cuando}\ \textit{AtD: LimpiarD}))$$

Los efectos condicionales no introducen indeterminación, pero pueden ayudar a modelarlo. Por ejemplo, supongamos que tenemos una tortuosa aspiradora que algunas veces descarga basura sobre el destino cuando se mueve, pero sólo si el destino está limpio. Este hecho puede ser modelado con una descripción del tipo

$$\textit{Acción}\ (\textit{Izquierda},\ \text{PRECOND}{:}\textit{AtD},\ \text{EFECTO}{:}\textit{AtI} \vee (\textit{AtI} \wedge \textbf{cuando}\ \textit{LimpiarI:}\ \neg\textit{LimpiarI})),$$

FASES CONDICIONALES

lo cual es tanto disyuntivo como condicional[6]. Para crear planes condicionales, necesitaremos **fases condicionales.** Las escribiremos usando la sintaxis «**si** <*evaluar*> **entonces** *plan_A* **y en caso contrario** *plan_B*», donde <*evaluar*> es una función booleana del estado de las variables. Por ejemplo, una fase condicional para el mundo aspiradora puede ser, «**si** $\textit{AtI} \wedge \textit{LimpiarI}$ **entonces** *Derecha* **y en caso contrario** *Aspirar*». La ejecución de tal fase prosigue de la manera obvia. Mediante la articulación de fases condicionales, los planes se transforman en árboles.

Queremos planes condicionales que trabajen *independientemente del resultado de la acción que ocurra*. Ya nos hemos topado con este problema antes, de un modo diferente. En juegos de dos-jugadores (Capítulo 6), queríamos que cada movimiento ganase *independientemente de cómo se moviese el oponente*. Por esta razón, los problemas de planificación no-determinística son frecuentemente llamados **juegos frente a naturaleza**.

JUEGOS FRENTE A NATURALEZA

Consideremos un ejemplo específico del mundo aspiradora. El estado inicial consiste en el robot situado en la posición de la derecha de un mundo limpio; como el entorno es completamente observable, el agente conoce la descripción completa del estado, $\textit{AtD} \wedge \textit{LimpiarI} \wedge \textit{LimpiarD}$. El estado objetivo consiste en el robot en la posición de la izquierda de un mundo limpio. Esto podría ser trivial, mientras no consideremos que la aspiradora algunas veces deposita suciedad cuando se mueve a una posición destino que se encuentre limpia, o que deposite algo de suciedad si *Aspirar* se aplica a una posición limpia.

Un «juego de tipo árbol» para este modelo de entorno se muestra en la Figura 12.9. Las acciones son seleccionadas por el robot en los «estados» nodo del árbol, y la naturaleza decide qué resultado ocurre en los nodos «oportunidad», representados como cír-

[6] El efecto condicional ***cuando*** *LimpiarI:*$\neg$ *LimpiarI* puede parecer un poco extraño. Recordemos, sin embargo, que aquí *LimpiarI* se refiere a la situación *previa* a la acción y $\neg$ *LimpiarI* se refiere a la situación *posterior* a la acción.

Figura 12.9 Los dos primeros niveles del árbol de búsqueda para el mundo aspiradora donde algunas veces deposite suciedad cuando se mueve. Los nodos estado son nodos OR donde alguna acción puede ser seleccionada. Los nodos oportunidad, representados como círculos, son nodos AND donde cada resultado puede ser manipulado, tal como indica el arco de unión de las ramas de salida. La solución se muestra en línea gruesa.

culos. Una solución será un sub-árbol que (1) tenga un nodo objetivo en cada «hoja», (2) especifique una acción en cada uno de sus nodos «estado», y (3) incluya cada rama resultado en cada uno de sus nodos «oportunidad». La solución se muestra en líneas gruesas en la figura; el grafo corresponde al plan [*Izquierda*, **si** $AtI \wedge LimpiarI \wedge LimpiarD$ **entonces** [ ] **en caso contrario** *Aspirar*]. (Hasta ahora, como estamos usando un planificador de espacio de estados, los tests en las fases condicionales consistirán en descripciones de estado completas.)

Para soluciones exactas de juegos, utilizaremos el **algoritmo minimax** (Figura 6.3). Para planificación condicional, tendremos dos modificaciones destacables. En primer lugar, los nodos MAX y MIN pueden convertirse en nodos OR y AND. Intuitivamente, el plan necesita tomar *alguna* acción en cada estado que alcanza, pero debe tratar con *todos* los resultados de la acción que tome. En segundo lugar, el algoritmo necesita retornar a un plan condicional preferible a sólo un único movimiento. En un nodo OR, el plan es únicamente la acción seleccionada, seguida por cualquier cosa que venga después. En un nodo AND, el plan es una articulación de cadenas «si-entonces-en caso contrario» especificando subplanes para cada resultado; los tests en estas fases son las descripciones completas de los estados[7].

Formalmente hablando, el espacio de búsqueda que hemos definido es un **grafo** AND-OR. En el Capítulo 7, los grafos AND-OR se ponen de manifiesto mediante cláusulas de inferencia de Horn. Aquí, las ramas del árbol son acciones más que etapas de

[7] Dichos planes podrían ser descritos también usando **casos**.

inferencia lógica, pero el algoritmo es el mismo. La Figura 12.10 proporciona un algoritmo recursivo de búsqueda en profundidad para un grafo de búsqueda AND-OR.

```
función GRAFO DE BÚSQUEDA AND-OR (problema) devuelve un plan condicional, o error
  BÚSQUEDA OR (ESTADO INICIAL [problema], problema, [ ])
---------------------------------------------------------------------------
función BÚSQUEDA OR (estado, problema, ruta) devuelve un plan condicional, o error
  si TEST-OBJETIVO[problema](estado) entonces devuelve el plan nulo
  si estado está en ruta entonces devuelve error
  para cada acción, conjunto_estados en SUCESORES [problema](estado) hacer
    plan ← BÚSQUEDA AND (conjunto_estados, problema,[estado | ruta])
    si plan ≠ error entonces devuelve [acción | plan]
  devuelve error
---------------------------------------------------------------------------
función BÚSQUEDA AND (conjunto_estados, problema, ruta) devuelve un plan condicional,
o error
  para cada s_i en conjunto_estados hacer
    plan_i ← BÚSQUEDA OR(s_i, problema, ruta)
    si plan = error entonces devuelve error
  devuelve [si s_1 entonces plan_1 en caso contrario si s_2 entonces plan_2 …
  si s_(n-1) entonces plan_(n-1) en caso contrario plan_n]
```

Figura 12.10 Un algoritmo para buscar grafos AND-OR generados por entornos no deterministas. Suponemos que SUCESORES devuelve una lista de acciones, asociadas cada una de ellas con un conjunto de salidas posibles. El objetivo es encontrar un plan condicional que alcanza un estado meta en toda circunstancia.

Un aspecto clave del algoritmo es la forma en la que se enfrenta a ciclos, que frecuentemente aparecen en problemas de planificación no-determinista (por ejemplo, si alguna acción a veces no tiene efecto, o si un efecto no intencionado puede ser corregido). Si el estado actual es idéntico a otro estado de la solución que viene de la raíz, entonces devolverá un error. Esto no significa que *no* haya solución para el estado actual; simplemente nos dice que si *existe* una solución no-cíclica, debe poder alcanzarse desde la primera encarnación del estado actual, de forma que la nueva encarnación pueda ser descartada. De este modo, certificamos que el algoritmo termina en cada espacio de estado finito, pues cada ruta debe alcanzar un objetivo, un estado final, o un estado repetido. Notemos que el algoritmo no chequea si el estado actual es una repetición de un estado de alguna otra ruta desde la raíz. El Ejercicio 12.15 investiga esta situación.

Los planes devueltos por GRAFOS DE BÚSQUEDA AND-OR contienen etapas condicionales que evalúan la descripción del estado completo para decidir qué rama seleccionamos. En muchos casos, tendremos que contar con menos tests exhaustivos. Por ejemplo, el plan solución de la Figura 12.9 podría ser escrito sencillamente como [*Izquierda*, **si** *LimpiarI* **entonces** [ ] **en caso contrario** *Aspirar*]. Esto es porque el test único, *LimpiarI*, basta para dividir los estados del nodo-AND en dos conjuntos de semiacierto, de modo que después del test el agente conoce exactamente en qué estado se encuentra.

De hecho, una serie de tests «si-entonces-en caso contrario» de variables simples siempre basta para dividir un conjunto de estados en semi-aciertos, *gracias* a que el estado es completamente observable. Podríamos, por tanto, restringir los tests a variables simples, sin pérdida de generalidad.

Existe una complicación final que frecuentemente emerge en dominios no-deterministas: las cosas no siempre funcionan a la primera, y se tiene que intentar de nuevo. Por ejemplo, consideremos el problema de la aspiradora «triple Murphy», el cual (además de sus hábitos establecidos previamente) algunas veces fracasa en el movimiento ordenado, por ejemplo, *Izquierda* puede tener el efecto disyuntivo $AtI \vee AtD$, como en la Ecuación (12.1). Por tanto, ahora no se garantiza que el plan [*Izquierda*, **si** *LimpiarI* **entonces** [ ] **en caso contrario** *Aspirar*] funcione. La Figura 12.11 muestra parte del grafo de búsqueda; obviamente no hay más soluciones acíclicas, y el GRAFO DE BÚSQUEDA AND-OR puede terminar sin error. Existe, sin embargo, una **solución cíclica**, que consiste en permanecer intentando *Izquierda* hasta que funcione. Podemos expresar esta solución añadiendo una **etiqueta** para denotar algunas porciones del plan y utilizaremos la etiqueta más tarde en lugar de repetir de nuevo el plan. De este modo, la solución cíclica es

SOLUCIÓN CÍCLICA

ETIQUETA

[I_1: *Izquierda*, **si** *AtD* **entonces** I_1 **en caso contrario si** *LimpiarI* **entonces** [ ] **en caso contrario** *Aspirar*]

(Una mejor sintaxis para la parte del bucle en el plan podría ser «**mientras** *AtD* **hacer** *Izquierda*».) Las modificaciones necesarias para los GRAFOS DE BÚSQUEDA AND-OR vienen explicadas en el Ejercicio 12.16. La ejecución clave es que un bucle que en el espacio de estado regresa al estado *I* se traduce en un bucle en el plan que vuelve al punto donde el subplan para el estado *I* es ejecutado.

Tenemos entonces la habilidad para sintetizar planes complejos que se asemejan a programas, con bucles y estructuras condicionales. Desafortunadamente, esto bucles son, potencialmente bucles *infinitos*. Por ejemplo, en la representación de acciones del mundo «triple Murphy» nada dice que *Izquierda* vaya a tener éxito. Los planes cíclicos, por tanto, son menos deseables que los planes acíclicos, pero pueden ser considerados soluciones, siempre que cada «hoja» sea un estado objetivo y pueda ser alcanzable desde cualquier punto del plan.

Figura 12.11 Primer nivel del grafo de búsqueda para el mundo aspiradora «triple Murphy» donde hemos mostrado ciclos explícitamente. Todas las soluciones para este problema son planes cíclicos.

Planificación condicional en entornos parcialmente observables

La sección precedente se dedicaba a explicar los entornos completamente observables, con la ventaja de que los tests condicionales pueden evaluar cualquier situación y de manera segura obtenemos una respuesta. En el mundo real, la observabilidad parcial es mucho más común. En el estado inicial de problema de planificación parcialmente observable, el agente conoce sólo una cierta parte del estado actual. El camino más simple para modelar esta situación es decir que el estado inicial pertenece a un **conjunto de estados**; el conjunto de estados es una forma de describir el **estado de creencias** iniciales del agente[8].

Supongamos que un agente del mundo aspiradora conoce que se encuentra en la posición D*erecha* y que dicha posición se encuentra limpia, pero que no puede darse cuenta de la presencia o ausencia de suciedad en otras posiciones. Entonces *tan pronto como conozca esta situación* podría encontrarse en dos posibles estados: la posición de su *Izquierda* podría estar tanto limpia como sucia. Este estado de creencias es etiquetado con *A* en la Figura 12.12. Ésta muestra parte del grafo AND-OR para el mundo de aspiradora que llamaremos «recíproco doble Murphy», en el cual la suciedad puede encontrarse a la Izquierda tras el agente cuando éste abandone una posición limpia[9]. Si el mundo fuese completamente observable, el agente podría construir una solución cíclica de la forma «Moverse hacia la derecha e izquierda, aspirando suciedad siempre que aparezca, hasta que todas las posiciones están limpias y me encuentre en la posición de la izquierda». (*Véase* el Ejercicio 12.16.). Desafortunadamente, con sensores locales de suciedad, este plan no es ejecutable, porque el valor de verdad del test «ambas posiciones están limpias» no puede ser determinado.

Hechemos un vistazo a cómo es construido el grafo AND-OR. Partiendo del estado de creencia *A*, mostramos el resultado de moverse hacia la *Izquierda* (el resto de acciones no tienen sentido). Como el agente puede olvidar suciedad detrás de él, los dos posibles mundos iniciales se convierten en cuatro posibles mundos, como se muestra en *B* y *C*. Los mundos constituyen dos estados de creencias posibles inicialmente, clasificados por el sensor de información disponible[10]. En *B,* el agente conoce *LimpiarI*; en *C* conoce ¬*LimpiarI*. Desde *C*, limpiar la suciedad mueve al agente hacia *B*. Desde *B*, moverse hacia la *Derecha* podría o no dejar suciedad tras de él, de modo que tenemos de nuevo cuatro posibles mundos, divididos de acuerdo con el conocimiento del agente, *LimpiarD* (retorno *A*) o ¬*LimpiarD* (estado de creencia *D*).

Resumiendo, los entornos no-deterministas y parcialmente observables nos proporcionan un grafo AND-OR de estados de creencia. Los planes condicionales pueden ser encontrados, por lo tanto, usando exactamente el mismo algoritmo que en el caso completamente observable, llamado GRAFO DE BÚSQUEDA AND-OR. Otra forma de entender

[8] Estos conceptos son introducidos en la Sección 3.6 que el lector puede consultar antes de proseguir.
[9] Padres con niños pequeños encontrarán familiar este fenómeno.
[10] Nótese que no están clasificados porque haya suciedad tras el agente al moverse. La ramificación en el espacio de estados de creencias está causado por resultados de conocimiento alternativo, y no resultados físicos.

Figura 12.12 Parte del grafo AND-OR para el mundo aspiradora «doble Murphy recíproco», en el cual la suciedad puede a veces ser abandonada tras el agente cuando éste se mueve hacia una posición limpia. El agente no es sensible a suciedad en otras posiciones.

qué sucede es comprobar que el *estado de creencia del agente* es siempre completamente observable (siempre conoce lo que conoce). La resolución «estándar» de problemas completamente observables es simplemente un caso especial en el que cada estado de creencia es un conjunto de semi-aciertos que contiene exactamente un estado físico.

¿Hemos acabado? No, aún no. Necesitamos decidir cómo los estados de creencia deberían ser representados, cómo los sensores funcionan y cómo las descripciones de las acciones deberán ser escritas en este nuevo escenario.

Existen tres opciones básicas para los estados de creencia:

1. Conjuntos de descripciones de estado completas. Por ejemplo, el estado de creencias inicial de la Figura 12.12 es

$$\{(AtD \wedge LimpiarD \wedge LimpiarI), (AtD \wedge LimpiarD \wedge \neg LimpiarI)\}$$

   Esta representación es sencilla para trabajar con ella pero muy costosa: si existen n proposiciones booleanas definiendo el estado, entonces un estado de creencias puede contener $O(2^n)$ descripciones de estados físicos, cada uno de

orden $O(n)$. Estados de creencia exponencialmente grandes ocurrirán cada vez que el agente conozca solamente una fracción de las proposiciones (cuantas menos conozca, más posibles estados podría alcanzar).

2. Sentencias lógicas que capturan exactamente el conjunto de posibles mundos en el estado de creencias. Por ejemplo, el estado inicial puede ser escrito como

$$AtD \wedge LimpiarD$$

Obviamente, cada estado de creencia puede ser capturado exactamente por una única sentencia lógica; en este caso, podremos usar la disyunción de todas las descripciones de estado conjuntivos, aunque nuestro ejemplo muestre que pudiesen existir sentencias más compactas.

Un inconveniente de las sentencias lógicas es que, como existen muchas diferentes, las sentencias lógicamente equivalentes describan el mismo estado de creencia, tal estado evaluado en el algoritmo de búsqueda en grafos puede requerir demostración de teoremas generales. Por esta razón, preferimos una representación *canónica* para sentencias en las que cada estado de creencias se corresponda con únicamente una sentencia[11]. Tal clase de representación usa una secuencia de literales ordenados por nombre de la proposición, $AtD \wedge LimpiarD$ es un ejemplo. Esta es simplemente la representación del estado estándar bajo la hipótesis de **mundo-abierto** que presentamos en el Capítulo 11. No todas las sentencias lógicas pueden ser escritas de tal manera (por ejemplo, no hay forma de representar $AtI \vee LimpiarD$) pero pueden ser tratados bastantes dominios.

3. **Proposiciones de conocimiento** describiendo el conocimiento del agente. (*Véase* también la Sección 7.7.) Para el estado inicial tenemos,

$$K(AtD) \wedge K(LimpiarD)$$

En esta expresión, K se traduce por «conoce que» y $K(P)$ significa que el agente conoce que P es cierto[12]. Con proposiciones de conocimiento, utilizamos la **hipótesis del mundo-cerrado** (si una proposición de conocimiento no aparece en la lista, se asume que es incorrecto). Por ejemplo, $\neg K(LimpiarI)$ y $\neg K(\neg LimpiarI)$ están implícitas en la sentencia anterior, de forma que se captura el hecho siguiente: el agente es ignorante acerca del valor de verdad de *LimpiarI*.

Observamos que la segunda y la tercera opción son, a grandes rasgos, equivalentes; usaremos la tercera opción, las proposiciones de conocimiento, porque nos ofrece una descripción más vívida del sentido y porque ya conocemos cómo escribir descripciones STRIPS con la hipótesis del mundo-cerrado.

En ambas opciones, cada símbolo proposicional puede aparecer en uno de los siguientes modos: positivo, negativo y desconocido. Por lo tanto, hay exactamente 3^n posibles estados de creencia que pueden estar descritos de este modo. Entonces, el conjunto

[11] La representación canónica más conocida para un sentencia proposicional genérica es el **diagrama de decisión binario** o DDB (Bryant, 1992).

[12] Es la misma notación usada para agentes basados en circuitos mencionados en el Capítulo 7. Algunos autores lo usan para hacer entender «conoce si P es verdad». La traducción entre las dos representaciones es directa.

de los estados de creencias es el súper-conjunto (conjunto de subconjuntos) del conjunto de estados físicos. Existen 2^n estados físicos, de modo que tenemos 2^n estados de creencias, lejos de las 3^n, de forma que las opciones 2 y 3 son bastante restrictivas como representaciones de estados de creencias. Esto actualmente se cree que es inevitable, porque *cualquier esquema capaz de representar cada posible estado de creencia requerirá bits para representar cada uno en el peor de los casos*. Nuestro sencillo esquema requiere solamente $O(n)$ bits para representar cada estado de creencia, intercambiando expresividad por solidez. Afortunadamente, los esquemas están clausurados bajo actualización STRIPS: tan pronto como lo expongamos en axiomas STRIPS, y si comenzamos con un estado que representa creencias, todos los estados de creencia posteriores serán representables también.

SENSORES AUTOMÁTICOS

SENSORES ACTIVOS

ACCIONES SENSORIALES

Necesitamos explicar cómo funcionan los sensores. Tenemos dos opciones. Podemos tener **sensores automáticos**, que permiten, a cualquier instante, que el agente obtenga todas las percepciones asequibles. El ejemplo de la Figura 12.12 asume sensores automáticos de localización y suciedad local. Alternativamente, podemos mencionar los **sensores activos**, en los cuales las percepciones son obtenidas solamente mediante la ejecución de **acciones sensoriales** específicas tales como *EvaluarSuciedad* y *EvaluarLocalización*. Abordaremos el estudio de cada tipo de sensores por turnos.

Escribamos una descripción de una acción utilizando proposiciones de conocimiento. Supongamos que el agente se mueve a la *Izquierda* en el mundo del «doble Murphy recíproco», con sensores de suciedad automáticos y locales; de acuerdo con las reglas de este mundo, el agente podría o no dejar suciedad tras de sí en el caso de que la posición estuviese limpia. Como efecto *físico* podría ser *disyuntivo*; pero como un efecto de *conocimiento*, simplemente elimina el conocimiento del agente de *LimpiarD*. El agente también conocerá si *LimpiarI* es verdadero, de un modo o de otro, gracias a los sensores de suciedad local, y conocerá que supone *AtI*:

$$\begin{array}{l} \textit{Acción}(\textit{Izquierda}, \text{PRECOND}{:}\textit{AtD}, \\ \quad \text{EFECTO}{:}\ K(\textit{AtI}) \wedge \neg K(\textit{AtD}) \wedge \textbf{cuando}\ \textit{LimpiarD}{:}\ \neg K(\textit{LimpiarD}) \wedge \\ \qquad \textbf{cuando}\ \textit{LimpiarI}{:}\ K(\textit{LimpiarI}) \wedge \\ \qquad \textbf{cuando}\ \neg\textit{LimpiarI}{:}\ K(\neg\textit{LimpiarI})) \end{array} \tag{12.2}$$

Destaquemos que tanto las precondiciones como las condiciones **cuando**, son proposiciones simples, no proposiciones de conocimiento. Esto es tal como debía ser porque los resultados de las acciones dependen del mundo actual, pero ¿cómo chequear la certeza de aquellas condiciones cuando todo lo que tenemos es un estado de creencia? Si el agente *conoce* un proposición, digamos $K(AtD)$, en el estado de creencia actual, entonces la proposición debe ser verdadera en el estado físico actual, y ciertamente la acción es aplicable. Si el agente no conoce una proposición (por ejemplo, la condición **cuando** *LimpiarI*) entonces el estado de creencia debe incluir mundos en los cuales *LimpiarI* sea verdad y mundos en los cuales *LimpiarI* sea falso. Esto hace crecer múltiples estados de creencias resultantes de las acciones. De este modo, si el estado inicial es $(K(AtD) \wedge K(LimpiarD))$, después de moverse hacia *Izquierda*, los dos estados de creencias resultantes son $(K(AtI) \wedge K(LimpiarI))$ y $(K(AtI) \wedge K(\neg LimpiarI))$. En ambos casos, el valor de verdad de *LimpiarI* es conocido, de forma que el test *LimpiarI* puede ser usado en el plan.

Con sensores activos (considerándolos opuestos a los sensores automáticos), el agente adquiere nuevas percepciones sólo consultándolas. De este modo, después de moverse hacia *Izquierda,* el agente no conocerá si la posición a mano izquierda está sucia, de modo que los dos efectos condicionales no aparecerán más en la descripción de la acción de la Ecuación (12.2). Para conocer si la posición está sucia, el agente puede *ChequearSuciedad:*

$$\begin{aligned}
&\textit{Acción}\ (\textit{ChequearSuciedad},\\
&\quad \textsc{Efecto}:\ \textbf{cuando}\ \textit{AtI} \wedge \textit{LimpiarI}: K(\textit{LimpiarI}) \wedge\\
&\qquad\qquad \textbf{cuando}\ \textit{AtI} \wedge \neg\textit{LimpiarI}: K(\neg\textit{LimpiarI}) \wedge\\
&\qquad\qquad \textbf{cuando}\ \textit{AtD} \wedge \textit{LimpiarD}: K(\textit{LimpiarD}) \wedge\\
&\qquad\qquad \textbf{cuando}\ \textit{AtD} \wedge \neg\textit{LimpiarD}: K(\neg\textit{LimpiarD}))
\end{aligned} \tag{12.3}$$

Es fácil probar que *Izquierda* seguido por *ChequearSuciedad* en el dispositivo de sensores activos da como resultado los mismos dos estados de creencia que ejecutar *Izquierda* con sensores automáticos. Con sensores activos, existe siempre el caso de acciones físicas que hacen corresponder a un estado de creencia un único estado de creencia sucesora. Múltiples estados de creencia pueden ser introducidos solamente por acciones sensoriales, que proporcionan conocimiento específico y por lo tanto permiten tests condicionales que pueden ser usados en los planes.

Hemos descrito un enfoque general para la planificación condicional basada en búsquedas AND-OR sobre el espacio de estados. Este enfoque ha demostrado ser bastante eficiente en algunos problemas, salvo algunos problemas intratables. Teóricamente, se puede mostrar que la planificación condicional pertenece a un tipo de complejidad mayor que la de la planificación clásica. Remarcamos que la definición de la clase NP es como sigue: una solución candidata puede ser chequeada para ver si es realmente una solución en tiempo polinomial. Esto es correcto para planes clásicos, de modo que el problema de la planificación clásica es un *NP*. Pero en planificación condicional, un candidato debe ser chequeado para comprobar si, para *todos* los posibles estados, existe *alguna* ruta a lo largo del plan que satisfaga el objetivo. Evaluar la combinación «todos/algunos» no puede ser hecho en tiempo polinomial, de forma que la planificación condicional es más costosa que *NP*. La única salida es ignorar algunas de las posibles contingencias durante la fase de planificación y tratarlas sólo cuando ellas realmente ocurran. Este es el enfoque que expondremos en la siguiente sección.

12.5 Vigilancia de ejecución y replanificación

Un agente de **vigilancia de ejecución** chequea sus percepciones para comprobar si todo va de acuerdo con el plan. La ley de Murphy nos dice que incluso los mejores planes humanos y los agentes de planificación condicional frecuentemente fracasan. El problema está indeterminado ilimitadamente (algunas circunstancias no anticipadas siempre surgirán para las cuales las descripciones de las acciones de los agentes no son correctas). Por tanto, la vigilancia de la ejecución es necesaria en entornos reales. Consideraremos dos tipos de vigilancia de ejecución: uno simple, pero débil, llamado

vigilancia de acciones, por medio de la cual el agente chequea el entorno para verificar que la siguiente acción funcionará, y otro más complejo pero más efectivo llamado **vigilancia de planes**, en el cual el agente verifica el plan que permanece de un modo global.

Un agente de **replanificación** conoce qué hacer cuando algo inesperado sucede: invocar un plan de nuevo para disponer de un nuevo plan que le permita alcanzar el objetivo. Para evitar consumir demasiado tiempo planificando, normalmente se intenta reparar el plan viejo para encontrar el modo en el que desde el estado actual inesperado se vuelva a disponer de un plan.

Como ejemplo, recordemos el mundo aspiradora del doble Murphy de la Figura 12.9. En este mundo, moverse a posiciones limpias, a veces, supone depositar suciedad en tales sitios; pero ¿y si el agente no conoce o no le preocupa este hecho? En ese caso, la solución es muy simple: [*Izquierda*]. Si no existe suciedad descargada cuando el plan está siendo ejecutado, el agente detectará que se alcanza un objetivo. En caso contrario, como la precondición *LimpiarI* de la etapa *Final* no está satisfecha, el agente generará un nuevo plan: [*Aspirar*]. La ejecución de este plan siempre es exitosa.

Juntos, la vigilancia de ejecución y la replanificación forman una estrategia general que puede ser aplicada tanto a entornos observables completamente como parcialmente, y con amplia variedad de representaciones de replanificación incluyendo espacio de estados, orden parcial y planes condicionales. Un esquema simple de planificación en el espacio de estados se muestra en la Figura 12.13. El agente de planificación comien-

```
función AGENTE-DE-REPLANIFICACIÓN (percepción) devuelve una acción
  estático: KB, base de conocimiento (incluye descripciones de acciones)
            plan, un plan, inicialmente [ ]
            plan_completo, un plan, inicialmente [ ]
            objetivo, un objetivo

  DECIR (KB, GENERAR-PERCEPCIÓN-SENTENCIA (percepción, t))
  actual ← DESCRIPCIÓN-ESTADO (KB, t)
  si plan = [ ] entonces
    plan_completo ← plan ← PLANNER (actual, objetivo, KB)
  si PRECONDICIONES(PRIMERO(plan)) no actualmente verdadero en KB entonces
    candidatos ← ORDENAR(plan_entero, ordenado por distancia a actual)
    encontrar un estado s en candidatos de modo que
      error ≠ reparación ← PLANNER (actual, s, KB)
    continuación ← poner a la cola de plan_completo comenzando en s
    plan_completo ← plan ← ADJUNTAR (reparación, continuación)
  devuelve POP(plan)
```

Figura 12.13 Agente de replanificación y vigilancia de ejecución. Utiliza un algoritmo de planificación de espacio de estados llamado PLANNER como una subrutina. Si las precondiciones de la siguiente acción no son apropiadas, el agente inicia bucles a partir de *p* puntos posibles en el *plan_completo*, intentando encontrar una ruta que PLANNER identifique como un plan. Llamamos a esta ruta *reparación*. Si PLANNER tiene éxito y encuentra un modo de repararlo, el agente adjunta *reparación* y lo pone a la cola del plan tras *p*, para construir el nuevo plan. El agente posteriormente regresa a la primera etapa en el plan.

za con un objetivo y crea un plan inicial para alcanzarlo. El agente entonces comienza ejecutando acciones una por una. El agente de replanificación, al contrario que otros agentes planificadores, sigue el rastro tanto del segmento del plan no ejecutado, *plan*, como del plan original y de forma completa, *plan_completo*. El agente utiliza **vigilancia de acciones**: antes de continuar con la siguiente acción del plan, el agente examina sus percepciones para ver si alguna de las precondiciones del plan no se ha satisfecho. Si es así, el agente intentará regresar a un plan mediante la replanificación de una secuencia de acciones que lo lleven de nuevo a un punto del *plan_completo*.

VIGILANCIA DE ACCIONES

La Figura 12.14 proporciona una ilustración esquemática del proceso. El replanificador notifica que las precondiciones de la primera acción en el *plan* no se encuentran satisfechas en el estado actual. Por tanto, se invoca al planificador para que genere un nuevo sub-plan llamado *reparación* que accederá desde la situación actual hasta algún estado *s* sobre el *plan_completo*. En este ejemplo, el estado *s* corresponde a una etapa anterior del actual *plan* (esto es así porque mantenemos la ruta del plan completo). En general, seleccionamos *s* para que esté lo más cercano posible al estado actual. La concatenación de la *reparación* y la porción del *plan_completo* de *s* en adelante, a lo que llamaremos continuación, genera un nuevo plan, y el agente se encuentra preparado para comenzar de nuevo la ejecución.

Figura 12.14 Antes de la ejecución, el planificador emerge con un plan, llamado *plan_completo,* para acceder a *G* desde *S*. El agente ejecuta el plan hasta el punto que se marca con *E*. Antes de ejecutar el *plan* que resta, se chequean las precondiciones y se encuentra que realmente hemos alcanzado el estado *O* y no el estado *E*. Se invoca al algoritmo de planificación para, mediante *reparación*, generar un plan que permita pasar del estado *O* a otro punto *P* del *plan_completo*. El nuevo *plan* se genera mediante la concatenación de *reparación* y *continuación* (la reanudación del original *plan_completo*).

Regresemos al ejemplo del problema de obtener una silla y una mesa del mismo color, esta vez vía replanificación. Se asume un entorno completamente observable. En el estado inicial la silla es azul, la mesa es verde, y se tiene una lata de pintura azul y otra de pintura roja. Esto nos lleva a la siguiente definición del problema:

$$\begin{aligned}&Iniciar\ (Color(Silla, Azul) \wedge Color(Mesa, Verde)\\ &\quad \wedge ContieneColor(BC, Azul) \wedge LataPintura(BC))\\ &\quad \wedge ContieneColor(RC, Rojo) \wedge LataPintura\ (RC)\end{aligned}$$

Objetivo (*Color* (*Silla, x*) ∧ *Color*(*Mesa, x*))
Acción(*Pintar*(*objeto, color*),
PRECOND: *TenerPintura* (*color*)
EFECTO: *Color*(*objeto, color*))
Acción (*Abrir*(*Lata*),
PRECOND: *LataPintura*(*Lata*) ∧ *ContieneColor*(*lata, color*)
EFECTO: *TenerPintura* (*color*)

El PLANNER del agente debe surgir con el siguiente plan:

[*Iniciar*; *Abrir*(*BC*); *Pintar*(*Mesa, Azul*); *Finalizar*]

Ahora el agente está preparado para ejecutar el plan. Se asume que todo saldrá bien cuando el agente abra el bote de pintura azul y aplique la pintura a la mesa. Los agentes de las secciones previas podrían declararse victoriosos en este momento, al haber completado las etapas del plan. Pero un agente con capacidad para la vigilancia de ejecución debe primeramente chequear la precondición de la etapa *Finalizar*, que implica comprobar que ambas piezas tienen al final el mismo color. Supongamos que el agente perciba que no comparten el mismo color, porque falta un pequeño trozo por pintar de la mesa. El agente entonces necesita calcular una posición en el *plan_completo* para apuntalar una secuencia de acción que lo repare. El agente conoce que el estado actual es idéntico a la precondición previa a la acción *Pintar*, de forma que el agente selecciona la secuencia vacía como *reparación* y transforma el *plan* en la misma secuencia [*Pintar, Finalizar*] que había intentado. Con este nuevo plan en juego, se reanuda la vigilancia de la ejecución, y la acción *Pintar* es reintentada. Este comportamiento será cíclico hasta que la mesa se perciba completamente pintada. Pero es importante destacar que el bucle es creado por un proceso de planificar-ejecutar-replanificar, más que por un bucle explícito en un plan.

La vigilancia de acciones es un método muy simple de vigilancia de ejecución pero a veces puede dirigirnos a comportamientos poco inteligentes. Por ejemplo, supongamos que el agente construye un plan para resolver el problema de pintar la silla y la mesa de rojo. Al abrir la lata de pintura roja comprueba que en ella sólo hay pintura para pintar la silla. La vigilancia de acciones podría no detectar errores hasta que la silla ya ha sido pintada, por lo que en ese instante *TenerPintura* (*rojo*) se convierte en falso. Lo que realmente necesitamos hacer es detectar un error en cualquier instante en que el estado sea tal que el plan que resta no funcione más. La **vigilancia del plan** permite esto, mediante el chequeo de las precondiciones para alcanzar el éxito en el plan que resta, es decir, las precondiciones de cada etapa en el plan, exceptuando aquellas que son alcanzadas por alguna otra etapa en el plan que resta por ejecutarse. La vigilancia de planes interrumpe tan pronto como es posible la ejecución de un bucle en un plan y no continúa hasta que el error ya haya ocurrido[13]. En algunos casos, esto puede rescatar al agente del desastre cuando el plan ha caído en un bucle y dirige al sistema a un punto muerto desde el que el objetivo podría ser inalcanzable.

VIGILANCIA DEL PLAN

[13] La vigilancia de planes logra un agente más listo que el escarabajo del estiércol (*véase* Apartado 2.2). Nuestro agente puede darse cuenta de que la bola de estiércol que ocultaba su guarida ya no está y podría replanificar para buscar otra nueva.

Es relativamente sencillo modificar un algoritmo de planificación para que permita al plan, en cada instante, tomar registro de las precondiciones para el éxito del plan que resta por ejecutarse. Si extendemos la vigilancia del plan a la evaluación de si el estado actual satisface las precondiciones del plan en cualquier instante del futuro, la vigilancia del plan podrá también extraer ventaja de la **serendipia**, esto es, del éxito accidental. Si alguna cosa sucede y la mesa se pinta de rojo al mismo tiempo que el agente está pintando la silla de rojo, las precondiciones del plan final se satisfacen (el objetivo ha sido alcanzado), y el agente puede regresar a casa más temprano.

Hasta aquí, hemos descrito la vigilancia y la replanificación en entornos completamente observables. Los procesos pueden volverse mucho más complicados cuando trabajamos con entornos parcialmente observables. En primer lugar, las cosas pueden estropearse sin que el agente sea capaz de detectarlo. En segundo lugar, «la evaluación de precondiciones» podría exigir la ejecución de acciones sensoriales, que también tienen que ser planificadas (en tiempo de planificación, lo que nos lleva de nuevo a planificación condicional, o respuestas en tiempo de ejecución). En el peor de los casos, la ejecución de una acción sensorial podría requerir un plan complejo que exigiese vigilancia y posiblemente más acciones sensoriales, etc. Si el agente insiste en chequear cada precondición, podría permanecer inactivo permanentemente y no hacer nada. Es preferible que el agente evalúe aquellas variables que son importantes, tenga buena suerte, y no emplee mucho tiempo en percibir fallos. Esto le permite al agente responder apropiadamente a situaciones de amenaza, pero que no malgaste el tiempo comprobando que «el cielo se desploma».

Hasta ahora hemos descrito un método de vigilancia y replanificación, pero la pregunta es «¿Funciona?» Esta es una pregunta con truco. Si se pretende decir «¿Se puede garantizar que el agente siempre alcanza el objetivo, incluso en sistemas con indeterminación ilimitada?» la respuesta es no, porque el agente podría alcanzar un punto muerto, tal como se describió en la Sección 4.5 en el ejemplo de la búsqueda *online*. Por ejemplo, el agente aspiradora podría no conocer que su batería puede acabarse. Esto descarta puntos muertos; es decir, supongamos que el agente puede construir un plan para alcanzar un objetivo desde *cualquier* estado en el entorno. Si suponemos que el entorno es no-determinista, en el sentido que dado un plan siempre tiene alguna posibilidad de éxito para cualquier tentativa de ejecución dada, entonces el agente eventualmente podrá alcanzar el objetivo. El agente de replanificación, por lo tanto, tiene capacidades análogas a los agentes de planificación condicional que hemos presentado. De hecho, podemos modificar un planificador condicional de modo que diseñe sólo un plan para una solución parcial y que incluya etapas de la forma «**si** <test> **entonces** *plan_A* **y en caso contrario** *replanificar*». Bajo las hipótesis planteadas, un plan de este tipo puede ser una solución correcta al problema original; además puede ser mucho más barato este diseño que un plan completamente condicional.

Hay problemas cuando el agente hace intentos repetidos para alcanzar el objetivo y son inútiles (cuando se bloquean por algunas precondiciones o efectos acerca de los cuales no se conoce). Por ejemplo, si el agente tiene la clave de la habitación de su hotel equivocada, no es posible abrir la puerta por muchas veces que inserte[14] la tar-

[14] Repeticiones inútiles de plan reparación exhiben el mismo comportamiento que la avispa sphex (*véase* Apartado 2.2).

jeta que abre su puerta. Una solución es seleccionar aleatoriamente uno de los elementos del conjunto de los planes de reparación posibles, preferiblemente al intento sucesivo del mismo. En este caso, el plan reparación que consiste en regresar a la recepción y solicitar una clave correcta de la habitación podría ser una alternativa útil. Dado que el agente puede no ser capaz de distinguir entre el caso realmente no-determinista y el caso inútil, es una buena idea aceptar las variaciones en las estretegias de reparación.

Otra solución para el problema de la descripción incorrecta de acciones es el **aprendizaje**. Después de varios intentos, el aprendizaje en agentes debe ser capaz de modificar la descripción de la acción que dice que la llave abre la puerta. En este instante, el replanificador automáticamente surgirá con un plan alternativo, como obtener una nueva llave. Esta clase de aprendizaje es descrita en el Capítulo 21.

Incluso con todas estas mejoras potenciales, el agente de replanificación aún tiene limitaciones. No puede comportarse correctamente en entornos a tiempo-real, y no hay límite sobre la cantidad de tiempo que empleará en replanificar, y por tanto en decidir la acción. Además, no se pueden formular sus nuevos objetivos añadidos a los que actualmente contempla, por lo que no vivirá mucho en un entorno complejo. Estas limitaciones se estudiarán en la sección siguiente.

12.6 Planificación continua

En esta sección, se diseñará un agente que persiste indefinidamente en un entorno. De forma que no tendremos un «resolvedor de problemas» sino que dado un objetivo, se ejecutarán planes y acciones hasta que el objetivo sea alcanzado; mejor dicho, el agente vive a través de una serie de formulaciones de objetivos que continuamente cambian, planificación y fases de actuación. Más que pensar en un monitor planificador y en un ejecutor de órdenes como procesos separados, uno de los cuales se basa en los datos que proporciona el otro, podemos considerarlos como un único proceso de un **agente de planificación continua**.

AGENTE DE PLANIFICACIÓN CONTINUA

Se viene considerando a los agentes como los *elementos a través de los que se ejecuta* un plan (el gran plan de vivir su vida). Sus actividades incluyen realizar algunas etapas del plan que están preparadas para ser ejecutadas, refinando el plan para satisfacer precondiciones abiertas o resolver conflictos, y modificando el plan a la luz de información adicional obtenida durante la ejecución. Obviamente, cuando se formula un nuevo objetivo, el agente no tendrá acciones preparadas para ejecutarlas, de forma que tardará un tiempo en la generación de un plan parcial. Los agentes de planificación continua vigilan el mundo continuamente, actualizando su modelo del mundo con nuevas percepciones aunque se mantengan las deliberaciones sobre objetivos.

Primeramente introduzcamos un ejemplo, y después describamos el programa del agente. Se utilizarán planes de orden-parcial como representaciones y llamaremos al resultado algoritmo POP-CON. Para simplificar la presentación, se supone un entorno completamente observable. Las mismas técnicas pueden ser extendidas para el caso parcialmente observable.

Figura 12.15 Secuencia de estados del agente de planificación continua que intenta alcanzar el estado objetivo *Sobre*(*C, D*) ∧ *Sobre*(*D, B*), que se muestra en (d). El estado inicial es (a). En (b), otro agente ha interferido en el plan, colocando *D* sobre *B*. En (c), el agente ha ejecutado *Mover*(*C, D*) pero ha fallado, y ha colocado *C* sobre *A*. En la última fase reintenta *Mover*(*C, D*), alcanzando el estado objetivo (d).

El ejemplo que presentaremos es un problema en el dominio del mundo de los bloques (Sección 11.1). El estado inicial se muestra en la Figura 12.15(a). La acción que necesitaremos es *Mover* (*x, y*), la cual mueve el bloque *x* sobre el bloque *y*, en el caso que ambos estén libres de obstáculos. El esquema de acción es:

Acción(Mover (x, y),
PRECOND: *SinObstáculos*(*x*) ∧ *SinObstáculos*(*y*) ∧ *Sobre*(*x, z*),
EFECTO: *Sobre*(*x, y*) ∧ *SinObstáculos*(*z*) ∧¬*Sobre*(*x, z*) ∧ ¬*SinObstáculos* (*y*))

El agente primeramente necesita formular un objetivo por él mismo. No se discutirá la formulación de objetivos en este momento, pero en su lugar asumiremos que de alguna forma el agente está informado para alcanzar el objetivo *Sobre*(*C, D*) ∧ *Sobre*(*D, B*). El agente comienza planificando para lograr este objetivo. Diferente a todo el resto de nuestros agentes, que interrumpen sus percepciones hasta que el planificador obtiene una solución completa al problema, el agente de planificación continua va construyendo el plan de un modo incremental, y cada incremento toma una limitada porción de tiempo. Después de cada incremento, el agente propone *NoOp* como acción y evalúa su percepción de nuevo. Se supone que las percepciones no cambian y que el agente construye el plan rápidamente. Éste se presenta en la Figura 12.16. Destacamos que aunque las precondiciones de ambas acciones se satisfacen por *Iniciar*, existe una restricción de orden entre *Mover*(*D, B*) antes de *Mover*(*C, D*). Esto es necesario para asegurar que *SinObstáculos* (*D*) es correcto hasta que *Mover* (*D, B*) se haya completado. A lo largo del proceso de

Figura 12.16 Plan inicial construido por el agente de planificación continua. El plan es indistinguible, hasta este momento, del que produjese un planificador normal de orden parcial.

planificación continua, *Iniciar* siempre usa etiqueta del estado presente. El agente actualiza el estado después de cada acción.

El plan está ahora preparado para ser ejecutado, pero antes de que el agente pueda tomar ninguna acción, la naturaleza interviene. Un agente externo (quizás el tutor del agente se empieza a impacientar) mueve *D* sobre *B* y el mundo se encuentra entonces en el estado mostrado en la Figura 12.15(b). El agente percibe esto, reconoce que *SinObstáculos*(*B*) y *Mover*(*D, G*) no son verdaderas en el estado actual, y actualiza su modelo de estado actual de forma correspondiente. Los lazos causales que proporcionaron las precondiciones *SinObstáculos*(*B*) y *Mover*(*D, G*) para la acción *Mover*(*D, B*) se han convertido en no-válidas y deben ser eliminadas del plan. El nuevo plan se muestra en la Figura 12.17. En todo este tiempo, *Iniciar* representa el estado actual, de forma que este *Iniciar* es diferente al mostrado en la figura anterior. Destacamos que el plan está aún incompleto: dos de las precondiciones para *Mover*(*D, B*) están abiertas, y su precondición *Sobre* (*D, y*) está no-instanciada, porque no existe ninguna razón para suponer que ese movimiento nos llevará a *G*.

Figura 12.17 Después de que algo moviese *D* sobre *B*, los enlaces que proporcionaron *SinObstáculos*(*B*) y *Mover*(*D, G*) son eliminados, produciendo este plan.

Ahora que el agente puede tomar ventaja de la «ayuda» de la interferencia, notificando que el enlace causal *Mover*(*D, B*) $\xrightarrow{\text{sobre } (D,B)}$ *Finalizar* puede ser reemplazado por un enlace directo desde *Iniciar* hasta *Finalizar*. Este proceso se conoce como **extensión** de un enlace causal y puede hacerse siempre que una condición propuesta por una etapa anterior o incluso posterior pueda provocar un nuevo conflicto.

EXTENSIÓN

Una vez que el antiguo lazo causal desde *Mover*(*D, B*) hasta *Finalizar* es eliminado, *Mover*(*D, B*) no proporciona más enlaces causales de ningún tipo. Se conoce como una **fase redundante**. Todas las fases redundantes, y cualquiera de los enlaces que la proporcionan, son eliminados del plan. Esto nos da el plan de la Figura 12.18.

FASE REDUNDANTE

Figura 12.18 El enlace que nos proporciona *Mover*(*D, B*) ha sido reemplazado por otro *Iniciar*, y la fase redundante *Mover*(*D, B*) ha sido eliminada.

Ahora la etapa *Mover(C, D)*, está preparada para ser ejecutada, porque todas sus precondiciones son satisfechas por la fase *Iniciar,* ninguna otra etapa está necesariamente antes, y no existe conflicto con cualquier otra etapa en el plan. Esta fase es eliminada del plan. Desafortunadamente, el agente es torpe y coloca *C* sobre *A* y no sobre *D*, proporcionando el estado mostrado en la Figura 12.15(c). El nuevo estado del plan se muestra en la Figura 12.19. Destaquemos que aunque no existan acciones en el plan en este momento, aún existe una condición abierta para la etapa *Finalizar*.

Figura 12.19 Después de que *Mover*(*C, D*) se ejecute y elimine del plan, los efectos de la fase *Iniciar* reflejan el hecho de que *C* termine sobre *A* en lugar de sobre *D*. La precondición del objetivo *Sobre*(*C, D*) permanece abierta.

El agente decide planificar a partir de condiciones abiertas. De nuevo, *Mover*(*C, D*) satisface una condición objetivo. Sus precondiciones son satisfechas por nuevos enlaces causales desde la fase *Iniciar*. El nuevo plan se presenta en la Figura 12.20.

Figura 12.20 La condición abierta se resuelve añadiendo *Mover*(*C, D*) de nuevo. Destacamos los nuevos lazos entre precondiciones.

De nuevo *Mover*(*C, D*), está preparado para ejecutarse. Esta vez sí funciona, obteniendo como resultado el estado objetivo mostrado en la Figura 12.15(d). Una vez que esta etapa ha sido superada en el plan, la condición objetivo *Sobre*(*C, D*) se convierte de nuevo en condición abierta. Aunque la fase *Iniciar* está actualizada para que represente el nuevo estado del mundo, sin embargo, la condición objetivo puede ser satisfecha inmediatamente por un enlace desde la fase *Iniciar*. Este es el curso de sucesos normales cuando una acción es exitosa. El estado final del plan se muestra en la Figura 12.21. Como todas las condiciones objetivo son satisfechas por la fase *Iniciar* y no quedan acciones pendientes, el agente es ahora libre de eliminar los objetivos de *Finalizar* y formular un nuevo objetivo.

Con este ejemplo, podemos ver que la planificación continua es bastante similar a la planificación de orden-parcial. Para cada iteración, el algoritmo encuentra alguna cosa acerca del plan que necesita reparación (un **defecto del plan**) y lo soluciona. El algoritmo POP puede ser visto como un algoritmo extractor-de-defectos, considerando defectos tanto las

DEFECTO DEL PLAN

Figura 12.21 Después de que *Mover*(*C, D*) sea ejecutado y eliminado del plan, la concisión abierta que permanece *Sobre*(*C, D*) se resuelve añadiendo un enlace causal desde la nueva etapa *Iniciar*. El plan es definitivamente completado.

precondiciones abiertas como los conflictos causales. El agente de planificación continua, además, puede solucionar un rango mucho más amplio de defectos:

- *Objetivos perdidos:* el agente puede decidir añadir un nuevo objetivo al estado *Finalizar* (en la planificación continua, podría tener más sentido cambiar el nombre de *Finalizar* por *No-finalizado*, y cambiar *Iniciar* por *Actual*, pero utilizaremos los términos tradicionales).
- *Precondición abierta:* añadir un enlace causal a una precondición abierta, seleccionando una acción nueva o una ya existente (como en POP).
- *Conflictos causales:* dado un enlace causal $A \xrightarrow{p} B$, y una acción C con efecto $\neg p$, seleccionar una restricción de orden o de alguna variable para resolver el conflicto (como en POP).
- *Enlaces sin fundamento*: si existe un enlace causal $Iniciar \xrightarrow{p} A$ donde p no es verdadero en *Iniciar,* entonces eliminamos el enlace. (Esto evita que ejecutemos acciones cuyas precondiciones sean falsas.)
- *Acciones redundantes*: si una acción A proporciona enlaces no causales, se elimina la acción y sus enlaces, esto nos permite tener ventaja de la serendipia de los eventos en el entorno.
- *Acciones no-ejecutadas*: si una acción A (diferente de *Finalizar*) tiene sus precondiciones satisfechas en *Iniciar*, no tiene otras acciones (además de *Iniciar*) antes que ella, y entra en conflicto con enlaces no causales, entonces se elimina A y sus enlaces causales y se restituye cuando la acción vaya a ser ejecutada.
- *Objetivos históricos innecesarios*: si no existen precondiciones abiertas ni acciones en el plan (de forma que todos los enlaces causales van directamente desde *Iniciar* a *Finalizar*), entonces hemos alcanzado el conjunto actual de objetivos. Eliminar los objetivos y los enlaces entre ellos permite tener en cuenta nuevos objetivos.

El AGENTE-POP-CONTINUO se muestra en la Figura 12.22. Consta de un ciclo «percibir, eliminar defectos, actuar». Mantiene un plan persistente en su base de conocimiento, y en cada turno elimina un defecto del plan. Entonces selecciona y ejecuta una acción (aunque frecuentemente la acción será *NoOp*) y repite el bucle. Este agente puede manejar muchos tipos de problemas que se enumeran en la discusión sobre agentes de replanificación del Apartado 12.6. En particular, puede actuar en tiempo real, se aprovecha de la serendipia del entorno, puede formular sus propios objetivos y puede gestionar efectos inesperados que afecten a planes futuros.

```
función AGENTE-POP-CONTINUO (percepción) devuelve una acción
   estático: plan, un plan, inicialmente simplemente constituido por Iniciar, Finalizar

   acción ← NoOp (el defecto)
   EFECTOS[Iniciar] = ACTUALIZAR(EFECTOS[Iniciar], percepción)
   ELIMINAR-DEFECTO(plan) // posiblemente actualizando acción
   devuelve acción
```

Figura 12.22 El AGENTE-POP-CONTINUO, un agente de planificación continua de orden-parcial. Después de recibir una percepción, el agente elimina un defecto de su, continuamente actualizado, plan y propone una acción. Normalmente esto llevará muchas fases de planificación eliminando-defectos, durante las cuales se devuelva *NoOp,* antes de que esté preparado para ejecutar una acción real.

12.7 Planificación multiagente

Hasta el momento hemos trabajado con entornos de **agente-único**, en los que nuestro sistema se encuentra sólo. Cuando existen otros agentes en el entorno, nuestro agente podría simplemente incluirlos en su modelo del contexto, sin cambiar fundamentalmente sus algoritmos básicos. En muchos casos, esto podría llevarnos a un comportamiento inadecuado porque tratar con otros agentes no es lo mismo que tratar con la naturaleza. En particular, la naturaleza es (se supone) indiferente a las intenciones de los agentes[15], mientras que otros agentes no lo serán. Esta sección introduce la planificación multiagente para trabajar con estos temas.

Hemos visto en el Capítulo 2 que el entorno multiagente puede ser **cooperativo** o **competitivo**. Comenzaremos con un ejemplo cooperativo simple: planificación de equipos de dobles en el tenis. Los planes pueden ser construidos de modo que se especifiquen acciones para ambos jugadores del equipo; describiremos técnicas para el diseño de planes eficientemente. La construcción de planes eficientes es útil, pero no garantiza el éxito; los agentes tienen que estar de acuerdo en utilizar el mismo plan. Esto requiere alguna forma de **coordinación**, posiblemente alcanzada a través de la **comunicación**.

Cooperación: planes y objetivos conjuntos

Dos agentes jugando en equipos de dobles en el tenis tienen el objetivo conjunto de ganar el partido, lo que se convierte en varios sub-objetivos. Supongamos que en un instante concreto del juego, se tiene el objetivo conjunto de devolver la bola que ha sido golpeada por el otro equipo y se quiere asegurar que al menos uno de los dos jugadores esté cubriendo la red. Podemos representar esta noción como un problemas de **planificación multiagente**, como se ha mostrado en la Figura 12.23.

PLANIFICACIÓN MULTIAGENTE

[15] Los residentes de Reino Unido, donde el mero acto de organizar un picnic significa que lloverá, pueden estar en desacuerdo.

Agentes (*A*, *B*)
Iniciar(*En* (*A*, [*Izquierda*, *LíneaFondo*]) ∧ *En* (*B*, [*Derecha*, *Red*]) ∧
  Acercándose (*Bola*, [*Derecha*, *LíneaFondo*])) *Compañero* (*A*, *B*) ∧ *Compañero* (*B*, *A*)
Objetivo (*Devolver* (*Bola*) ∧ *En*(*agente*, [*x*, *Red*]))
Acción (*Golpear*(*agente*, *Bola*),
  PRECOND: *Acercándose* (*Bola*, [*x*, *y*]) ∧ *En*(*agente*, [*x*, *y*]) ∧
    Compañero (*agente*, *compañero*) ∧ ¬*En*(*compañero*, [*x*, *y*])
  EFECTO: *Devuelta*(*Bola*))
Acción (*Ir*(*agente*, [*x*, *y*]),
  PRECOND: *En*(*agente*, [*a*, *b*]),
  EFECTO: *En*(*agente*, [*x*, *y*]) ∧ ¬*En*(*agente*, [*a*, *b*]))

Figura 12.23 El problema del partido de tenis de dobles. Dos agentes están jugando juntos y pueden estar en una de las siguientes cuatro situaciones: [*Izquierda, LíneaFondo*], [*Derecha, LíneaFondo*], [*Izquierda, Red*] y [*Derecha, Red*]. La bola sólo puede ser devuelta si existe un jugador en el lugar correcto.

Esta notación incluye dos nuevos elementos. En primer lugar, la sentencia *Agentes*(*A*, *B*) nos dice que existen dos agentes, *A* y *B*, que participarán en el plan. (Para este problema los jugadores contrarios no son considerados agentes.) En segundo lugar, cada acción menciona explícitamente al agente como un parámetro, porque necesitamos mantener la pista de qué agente hace qué cosa.

PLAN CONJUNTO

Una solución al problema de planificación multiagente es un **plan conjunto** que consiste en acciones para cada uno de los agentes. Un plan conjunto es una solución si el objetivo es alcanzado cuando cada agente ejecute las acciones que se le han asignado. El siguiente plan es una solución para el problema del tenis:

PLAN 1:
  A: [*Ir*(*A*, [*Derecha*, *LíneaFondo*]),*Golpear*(*A*, *Bola*)]
  B: [*NoOp*(*B*), *NoOp*(*B*)]

Si ambos agentes tienen la misma base de conocimiento, y si ésta es la única solución, entonces puede ser correcto; los agentes podrían buscar cada uno la solución y ejecutarla conjuntamente. Desafortunadamente para los agentes (y pronto veremos por qué es desafortunado), existe otro plan que satisface el objetivo tan adecuadamente como el anterior:

PLAN 2:
  A: [*Ir*(*A*, [*Izquierda*, *Red*]),*NoOp*(*A*)]
  B: [*Ir*(*B*, [*Derecha*, *LíneaFondo*]),*Golpear*(*B*, *Bola*)]

Si *A* selecciona el plan 2 y *B* selecciona el plan 1, entonces ninguno devolverá la bola. Y al contrario, si *A* selecciona el plan 1 y *B* selecciona el plan 2, posiblemente colisionarán uno contra otro; ninguno devuelve la bola y la red puede quedar desprotegida. No debe extraerse de aquí que la existencia de planes conjuntos no signifique que el objetivo sea alcanzado. Los agentes necesitan un mecanismo de **coordinación** para alcanzar el *mismo* plan conjunto; y por tanto, deben tener conocimiento común (*véase* Capítulo 10) para que un plan conjunto en particular pueda ser ejecutado.

COORDINACIÓN

Planificación condicional en entornos parcialmente observables

PLANIFICACIÓN MULTICUERPO

Esta sección se concentra en la construcción de planes conjuntos correctos, aplazando la coordinación para más tarde. Llamamos a esto **planificación multicuerpo**; éste es esencialmente un problema de planificación afrontado por un agente centralizado y único que puede dictar acciones a varias entidades físicas. En el caso multiagente real, esto capacita a cada agente a tener en cuenta qué posibles planes conjuntos podrían tener éxito si se ejecutan conjuntamente.

Nuestro estudio de la planificación multicuerpo está basado en la planificación de orden-parcial, tal como se describió en la Sección 11.3. Supondremos observabilidad completa, para estudiar un caso más sencillo. Existe un elemento nuevo que no aparece en el caso de agente único: el entorno no es realmente **estático**, porque otros agentes podrían actuar mientras un agente en particular se encuentra deliberando. Por tanto, se debe tener en cuenta la **sincronización**. Para simplificar, supondremos que cada acción emplea la misma cantidad de tiempo y que las acciones en cada instante del plan conjunto son simultáneas.

SINCRONIZACIÓN

Para todo instante de tiempo, cada agente está ejecutando únicamente una acción (quizás incluyendo *NoOp*). Este conjunto de acciones concurrentes se conoce como **acción conjunta**. Por ejemplo, un acción conjunta en el problema del tenis (Apartado 12.7) con dos agentes A y B es $\langle NoOp(A), Golpear(B, Bola)\rangle$. Un plan conjunto consiste en un grafo de acciones conjuntas parcialmente ordenado. Por ejemplo, el Plan 2 para el problema del tenis puede ser representado como la siguiente secuencia de acciones conjuntas:

ACCIÓN CONJUNTA

$$\langle Ir(A, [Izquierda, Red]), Ir(B, [Derecha, LíneaFondo])\rangle$$
$$\langle NoOp(A), Golpear(B, Bola)\rangle$$

Se *puede* planificar utilizando el algoritmo POP, aplicado a un conjunto de posibles acciones conjuntas. El único problema es el tamaño de este conjunto: con 10 acciones y cinco agentes tendremos 10^5 acciones conjuntas. Puede ser tedioso especificar las precondiciones y los efectos de cada acción, e ineficiente si queremos planificar con un conjunto tan grande.

Una alternativa consiste en definir acciones conjuntas implícitamente, describiendo cómo cada acción individual interactúa con otras posibles acciones. Esto será más simple, porque muchas acciones son independientes de otras; necesitaremos tener enumeradas las pocas acciones que realmente interactúan. Esto puede hacerse incrementando las descripciones STRIPS o ADL usuales con un nuevo elemento: una **lista de acciones concurrentes**. Ésta es similar a las precondiciones de la descripción de una acción excepto porque en lugar de describirse en términos de variables de estado, se describen las acciones que deben o no deben ser ejecutadas concurrentemente. Por ejemplo, la acción *Golpear* puede ser descrita como sigue:

LISTA DE ACCIONES CONCURRENTES

$$\begin{aligned}
&Acción\ (Golpear(A, Bola),\\
&\quad \text{CONCURRENTE}: \neg Golpear(B, Bola)\\
&\quad \text{PRECOND}: Acercándose\ (Bola, [x, y]) \wedge En(A, [x, y])\\
&\quad \text{EFECTO}: Devuelta(Bola))
\end{aligned}$$

Aquí se tiene una limitación de concurrencia-prohibida y exige que durante la ejecución de la acción *Golpear*, no pueda existir otra acción *Golpear* por otro de los agentes. Se pueden requerir acciones concurrentes, por ejemplo para que ambos agentes lleven una nevera de bebidas frías al banquillo. La descripción para esta acción nos asegura que el agente *A* no puede ejecutar la acción *Llevar* a no ser que exista otro agente *B* que esté simultáneamente ejecutando *Llevar* y aplicado al mismo refrigerador:

Acción (*Llevar*(*A, frigorífico, aquí, allí*),
CONCURRENTE: *Llevar*(*B, frigorífico, aquí, allí*)
PRECOND: *En*(*A, aquí*) $\wedge$ *En*(*frigorífico, aquí*) $\wedge$ *Frigorífico* (*frigorífico*)
EFECTO: *En*(*A, allí*) $\wedge$ *En*(*frigorífico, aquí*) $\wedge$ $\neg$*En*(*A, aquí*)
$\wedge$ $\neg$ *En*(*frigorífico, aquí*)

Con esta representación, es posible crear un planificador que sea muy parecido al planificador de orden parcial POP. Tenemos tres diferencias:

1. Además de la relación de orden temporal $A \prec B$, permitiremos $A = B$ y $A \preccurlyeq B$, que significan respectivamente «concurrente» y «antes de la concurrencia».
2. Cuando una nueva acción ha requerido acciones concurrentes, se deben instanciar aquellas acciones, utilizando acciones nuevas o existentes en el plan.
3. La prohibición de acciones concurrentes es una fuente adicional de limitaciones. Cada restricción debe ser resuelta mediante la limitación de acciones que antes o después entren en conflicto.

Esta representación nos da un algoritmo equivalente a POP para dominios multicuerpo. Podríamos extender este enfoque con los refinamientos de los dos últimos capítulos (RJTs, observabilidad parcial, condicionales, vigilancia de ejecución y replanificación) pero esto va más allá del objetivo de este libro.

Mecanismos de coordinación

CONVENCIÓN

El método más simple por el cual un grupo de agentes puede asegurar un acuerdo sobre un plan conjunto es adoptar una **convención** previa a la articulación de la actividad conjunta. Una convención es una restricción sobre la selección de planes conjuntos, más allá de las restricciones básicas que el plan conjunto debe establecer si todos los agentes lo aceptan. Por ejemplo, la convención «golpea desde el lado del juez de silla» podría forzar a que los agentes seleccionasen el Plan 2, mientras que la convención «un jugador siempre se debe encontrar en la red», nos lleva a la elección del Plan 1. Algunas convenciones, tales como conducir por el lado correcto de la vía, son tan ampliamente aceptadas que se consideran **leyes sociales.** Los idiomas de diferentes comunidades humanas pueden también ser vistos como convenciones.

LEYES SOCIALES

Las convenciones, tal como las hemos descrito en el párrafo anterior, son específicas de un dominio y pueden ser implementadas mediante la restricción de descripciones de acciones que eviten violaciones de la misma convención. Un enfoque más general consiste en usar convenciones independientes-de-dominio. Por ejemplo, si cada agente ejecuta el mismo algoritmo de planificación multicuerpo con los mismos *inputs*, se puede

seguir la convención de ejecutar el primero de los planes conjuntos que sea viable, confiando en que el resto de los agentes lleguen a la misma elección. Una estrategia más robusta pero más costosa podría ser la de generar todos los planes conjuntos y posteriormente seleccionar aquel cuya representación impresa es la primera alfabéticamente.

Las convenciones pueden también surgir de procesos evolutivos. Por ejemplo, las colonias de insectos sociales ejecutan planes conjuntos muy elaborados, los cuales están facilitados por la estructura genética común de los individuos de la colonia. La conformidad puede también ser forzada a través del hecho de que la desviación de las convenciones reduce el éxito evolutivo, de forma que cualquier plan conjunto viable puede llegar a ser un equilibrio estable. Consideraciones similares se aplican al desarrollo del lenguaje humano, donde la clave no está en qué lenguaje debe hablar un individuo, sino que todos los individuos de una comunidad hablen el mismo lenguaje. Un ejemplo final aparece en el comportamiento grupal que presentan los pájaros. Se puede obtener una simulación razonable si cada **agente pájaro** ejecuta las siguientes tres reglas combinadas entre sí:

AGENTE PÁJARO

1. Separación: alejarse de sus vecinos cuando comienza a acercarse.
2. Cohesión: moverse hacia la posición promedio entre sus vecinos.
3. Alineamiento: moverse en el sentido de la orientación de sus vecinos.

COMPORTAMIENTO EMERGENTE

Si todos los pájaros ejecutan la misma política, el grupo exhibe el **comportamiento emergente** en el vuelo que simula un cuerpo pseudo-rígido con densidad aproximadamente constante y que no se dispersa en el tiempo. Al igual que con los insectos, no hay necesidad de que cada agente posea el plan conjunto que modela las acciones de otros agentes.

Típicamente, las convenciones son adoptadas para dar cobertura a un universo de problemas de planificación multiagente, más que desarrollar convenciones nuevas para cada problema. Esto puede llevarnos a poca flexibilidad y fracasos, similar a algunas situaciones en el problema del tenis como que la bola llegue aproximadamente equidistante entre ambos compañeros. En ausencia de una convención aplicable, los agentes pueden utilizar **comunicación** para alcanzar conocimiento común sobre un plan conjunto y viable. Por ejemplo, en el problema del partido de dobles, un jugador puede gritar «¡Mía!» o «¡Tuya!» para indicar un plan conjunto. Veremos con más detalle mecanismos de comunicación en el Capítulo 22, donde se observará que la comunicación no necesariamente lleva asociado un intercambio verbal. Por ejemplo, un jugador puede comunicarle a otro un plan conjunto preferible para él, mediante la ejecución de la primera parte del plan. En nuestro problema del tenis, si el agente *A* sube a la red, el agente *B* está obligado a regresar a la línea de fondo, porque el Plan 2 es el único plan conjunto que se inicia con el agente *A* subiendo a la red. Este modo de coordinación, a veces llamado **reconocimiento de planes**, funciona cuando una acción única (o una pequeña secuencia de acciones) es suficiente para determinar un plan conjunto sin ambigüedad.

RECONOCIMIENTO DE PLANES

El coste de asegurarse que el agente alcanza un correcto plan conjunto puede ser cargado sobre los diseñadores de los agentes o sobre los mismo agentes. En el primer caso, antes de que los agentes comiencen un plan, el diseñador del agente debe probar que las políticas y las estrategias sean útiles. Los agentes en este caso pueden ser reactivos en su entorno, ya que no necesitan tener modelos explícitos acerca de otros agentes. En el

segundo caso, los agentes son deliberativos; ellos deben proporcionar o, en su defecto, demostrar que sus planes son eficaces, tomando en consideración el razonamiento de otros agentes. Por ejemplo, en un entorno con dos agentes lógicos A y B, ambos podrían tener las siguientes definiciones:

$$\forall\, p, s \;\; Accesible\,(p, s) \Leftrightarrow ConocimientoComún\,(\{A, B\}, Alcanza\,(p, s, Objetivo))$$

Esto nos dice que en cualquier situación s, el plan p es un plan conjunto accesible si existe el conocimiento común entre los agentes de que p alcanzará el objetivo. Necesitamos más axiomas para establecer el conocimiento común de una **intención conjunta** que ejecute un plan conjunto *particular*; sólo entonces pueden los agentes comenzar a actuar.

INTENCIÓN CONJUNTA

Mecanismos de coordinación

CONFLICTO

No todos los entornos multiagente implican la existencia de agentes cooperativos. Los agentes con funciones de utilidad incompatibles entran en **conflicto** con el resto. Un ejemplo son los juegos de suma-cero entre dos jugadores, como el ajedrez. Vimos en el Capítulo 6 que un agente-jugador-de-ajedrez necesita considerar los posibles movimientos del oponente antes de actuar. Por tanto, un agente en un entorno competitivo debe (a) reconocer que existen otros agentes, (b) calcular algunos planes posibles del resto de agentes, (c) calcular cómo los planes de otros agentes interactúan con sus propios planes, y (d) decidir sobre la mejor acción en vista de estas interacciones. Por tanto, la competición, como la cooperación, requieren un modelo de los planes del otro agente. En el caso de la competición, no existe compromiso para elegir un plan conjunto.

La Sección 12.4 dibujó la analogía entre juegos y problemas de planificación condicional. El algoritmo de planificación condicional de la Figura 12.12 diseña planes que funcionan bajo la hipótesis del peor entorno posible, de modo que puede ser aplicado en situaciones competitivas donde el agente sólo se mueve por éxito y fracaso. Cuando al agente y a sus oponentes les preocupa el *coste* de un plan, entonces el minimax es apropiado. Existen trabajos sobre la combinación del minimax con métodos tales como la planificación POP o RJT, que van más allá del modelo de búsqueda en el espacio de estados utilizado en el Capítulo 6. Volveremos al tema de la competición en la Sección 17.6 donde se estudia la teoría de juegos.

12.8 Resumen

En este capítulo nos hemos referido a algunas de las complicaciones de la planificación y la actuación en el mundo real. Los puntos principales son los siguientes:

- Muchas acciones consumen **recursos**, como dinero, gas o materias primas. Es conveniente trabajar con estos recursos en términos de medidas numéricas más que intentar razonar acerca de cada moneda individual. Las acciones pueden generar y consumir recursos, y es normalmente poco costoso y eficaz evaluar planes parciales para que se satisfaga la limitación de recursos antes que intentar refinamientos mayores.

- El tiempo es uno de los recursos más importantes. Puede ser gestionado por algoritmos de programación especializados, o pueden integrarse módulos de programación de tareas en la planificación.
- La planificación basada en **redes jerárquicas de tareas** (RJTs) permite al agente tener sugerencias sobre el dominio por parte del diseñador en la forma de reglas de descomposición. Esto hace viable crear planes extensos que son requeridos por muchas aplicaciones en el mundo real.
- Los algoritmos de planificación estándar asumen información completa, correcta y determinista, en entornos completamente observables. Muchos dominios violan esta hipótesis.
- La información incompleta puede ser manejada mediante planificación que utilice acciones sensoriales para obtener la información que necesita. Los **planes condicionales** permiten al agente ser sensible al mundo durante la ejecución para decidir qué rama del plan seguir. En algunos casos, la **planificación sin sensores** puede ser usada para diseñar planes que funcionen sin la necesidad de percepción. Tanto la planificación sin sensores como la condicional pueden diseñar planes mediante búsquedas en el espacio de los **estados de creencia**.
- La información incorrecta surge de precondiciones insatisfechas para planes y acciones. La **vigilancia de ejecución** detecta violaciones de las precondiciones que pongan en peligro el éxito del plan completo.
- Un **agente de replanificación** utiliza vigilancia de ejecución, y corta y repara cuando es necesario.
- Un **agente de planificación continua** crea nuevos objetivos en el transcurso de su ejecución y reacciones en tiempo real.
- La **planificación multiagente** es necesaria cuando existen otros agentes en el entorno con los que cooperar, competir o coordinarse. La planificación **multicuerpo** construye planes conjuntos, usando una descomposición eficiente para las descripciones de acciones conjuntas, además de alguna forma de coordinación si dos agentes cooperativos están de acuerdo sobre qué plan conjunto ejecutar.

Notas bibliográficas e históricas

La planificación en tiempo continuo fue por primera vez tratada por Deviser (Vere, 1983). El enfoque de una representación sistemática del tiempo en los planes fue planteada por Dean *et al*. (1990) en el sistema Forbin. Nonlin+ (Tate y Whiter, 1984) y Sipe (Wilkins, 1988, 1990) pudo razonar acerca de la distribución de recursos limitados para varias fases del problema. O-Plan (Bell y Tate, 1985), un planificador RJT, tenía una representación uniforme para las limitaciones de tiempo y recursos. Además de la aplicación ya mencionada de Hitachi en el capítulo, O-Plan ha sido aplicado a la planificación de diseño de *software* en Price Waterhouse y la planificación del ensamblaje de ejes traseros en los coches de la marca Jaguar. Un gran número de sistemas de planificación-y-programación híbridos han sido destacados: Isis (Fox *et al*., 1982; Fox, 1990) ha sido usado para la programación de tareas en Westinghouse, Gari (Descotte y Latombe, 1985)

planificó el diseño y el ajuste de partes mecánicas, FORBIN se utilizó para control industrial, y NONLIN+ se usó en planificación logística naval.

Después de un inicial agotamiento de la investigación teórica en la década de los 80, la planificación temporal hizo su aparición cuando nuevos algoritmos y mayor poder de procesado hicieron posibles atacar aplicaciones prácticas. Los dos planificadores SAPA (Do y Kambhampati, 2001) y T4 (Haslum y Geffner, 2001) utilizaron búsquedas hacia-delante en el espacio de estados con sofisticadas heurísticas para trabajar con acciones y recursos. Una alternativa consiste en el uso de lenguajes expresivos, pero guiados por heurísticas en humanos y en dominios específicos, como se hizo en ASPEN (Fukunaga *et al*., 1997), HSTS (Jonson *et al*., 2000) e IXTET (Ghallab y Laruelle, 1994).

Hay una larga tradición en programación de tareas en el sector aeroespacial. T-SCHED (Drabble, 1990) se utilizó para programar los comandos de la misión del satélite UOSAT-II. OPTIMUM-AIV (Aarup *et al*., 1994) y PLAN-ERS 1 (Fuchs *et al*., 1990), ambos basados en O-PLAN, fueron usados para el ensamblaje de un vehículo espacial y la planificación, respectivamente, en la Agencia Espacial Europea. SPIKE (Johnston y Adorf, 1992) se utilizó por la NASA en la planificación del Telescopio Espacial Hubble, mientras que el Sistema de Programación de Procesado de Lanzadera Espacial (Deale *et al*., 1994) realizaba la programación de tareas de más de 16.000 trabajadores. El Agente Remoto (Muscettola *et al.,* 1998) se convirtió en el primer planificador autónomo para controlar una aeronave que voló sobre la sonda Deep Space One en 1999. La literatura sobre programación de tareas en investigación operativa fue supervisada por Vaessens *et al*. (1996); trabajos teóricos fueron presentados por Martin y Shmoys (1996).

MACROPS

La facilidad en programación STRIPS para el aprendizaje **macrops** («macro-operadores» que consisten en una secuencia de etapas primitivas) podría ser considerada el primer mecanismo para planificación jerárquica (Fikes *et al*., 1972). La jerarquía fue también utilizada en el sistema LAWALY (Siklossy y Dreussi, 1973). El sistema ABSTRIPS (Sacerdoti, 1974) introdujo la idea de **jerarquía abstracta**, por medio de la cual la planificación en niveles elevados era posible ignorando las precondiciones de acciones de niveles inferiores con el fin de obtener la estructura general del plan de trabajo. La tesis doctoral de Austin Tate (1975b) y el trabajo de Earl Sacerdoti (1977) fundamentaron las ideas básicas de la planificación RJT en su forma moderna. Varios planificadores prácticos, incluyendo O-PLAN y SIPE, son planificadores RJT. Yang (1990) discutió las propiedades de acción que hacían a los RJT eficientes. Erol, Hendler y Nau (1994, 1996) presentaron un planificador de descomposición jerárquica completa en un rango de complejidad mayor que los planificadores RJT puros. Otros autores (Ambros-Ingerson y Steel, 1988; Young *et al*., 1994; Barrett y Weld, 1994; Kambhampti *et al*., 1998) han propuesto un enfoque híbrido presentado en este capítulo, en el cual las descomposiciones son simplemente otra forma de refinamiento que puede ser usado en planificación de orden-parcial.

JERARQUÍA ABSTRACTA

Empezando con el trabajo con macro-operadores en STRIPS, uno de los objetivos de la planificación jerárquica fue la reutilización de experiencias de planificación previas en la forma de planes generalizados. La técnica de **aprendizaje basado en la explicación**, descrita en profundidad en el Capítulo 19, ha sido aplicado en varios sistemas como una forma de generalización de planes previamente computados, incluyendo SOAR (Laird *et al.,* 1986) y PRODIGY (Carbonell *et al*., 1989). Un enfoque alternativo es el almace-

naje de planes previamente computados en su forma original y posteriormente reutilizados para resolver nuevos problemas similares por analogía con el problema original. Este es el enfoque tomado por un área llamada **planificación basada en casos** (Carbonell, 1983; Alterman, 1988; Hammond, 1989). Kambhampati (1994) argumenta que la planificación basada en casos debería ser analizada como una forma de refinamiento en la planificación y propone un fundamento formal para la planificación basada en casos de orden parcial.

PLANIFICACIÓN BASADA EN CASOS

La imprevisibilidad y observabilidad parcial de entornos reales fue expuesta en proyectos de robótica que usaban técnicas de planificación, incluyendo Shakey (Fikes *et al*., 1972) y FREDDY (Michie, 1974). El problema recibió más atención después de la publicación del importante artículo de McDermott (1978a), *Planning y Acting*.

Los primeros planificadores, sin condicionales ni bucles, no pudieron reconocer explícitamente el concepto de planificación condicional; pero a pesar de todo emplearon un estilo coercitivo en respuesta a la incertidumbre del entorno. El sistema NOAH de Sacerdoti usaba la coerción en su solución al problema «llaves y cajas», un problema de planificación en el que el planificador conocía muy poco acerca del estado inicial. Mason (1993) argumentaba que la percepción sensorial frecuentemente podía y debía ser distribuida con un robot planificador, y describió un plan sin sensores que podía mover un elemento en una posición específica sobre una mesa mediante una secuencia de acciones de inclinación, *independientemente* de su posición original. Se describirá esta idea en el contexto de la robótica (*véase* Figura 25.17).

Goldman y Boddy (1996) introdujeron el término **planificación conforme** para planificadores sin sensores que tratasen la incertidumbre forzando que el mundo alcanzase estados conocidos, destacando que los planes sin sensores son efectivos incluso aunque el agente posea sensores. El primer planificador moderadamente eficiente fue el Conformant Graphplan (CGP) de Smith y Weld (1988). Ferraris y Giunchiglia (2000) y Rintanen (1999) desarrollaron de manera independiente planificadores conformes basados en SATplan. Bonet y Geffner (2000) describieron un planificador conforme basado en búsquedas heurísticas en el espacio de los estados de creencia, trabajando con ideas desarrolladas por primera vez en 1960 en los procesos de decisión de Markov parcialmente observables, o PDMPOs (*véase* Capítulo 17). Actualmente, los planificadores conformes de estado de creencias más eficaces, como HSCP (Bertoli *et al*., 2001a), usan diagramas de decisión binarios (BDDs) (Bryant, 1992) para representar estados de creencias y son del orden de cinco órdenes de magnitud más rápidos que el CGP.

WARPLAN-C (Warren, 1976), una variante de WARPLAN, fue uno de los primeros planificadores en usar acciones condicionales. Olawski y Gini (1990) diseñaron los mejores sistemas de planificación condicional.

El enfoque de la planificación condicional descrito en el capítulo está basado en los algoritmos de búsqueda para grafos AND-OR desarrollados por Jiménez y Torras (2000) y por Hansen y Zilberstein (2001). Bertoli *et al*., (2001b) describieron un planteamiento basado en BDD que construía planes condicionales con bucles. C-BURIDAN (Draper *et al.,* 1994) trata la planificación condicional para acciones con resultados probabilísticos, un problema también enfocado por los sistemas POMDPs (Capítulo 17).

Existe una relación próxima entre la planificación condicional y la síntesis de programas automáticos; sus referencias aparecen en el Capítulo 9. Ambas áreas han sido

estudiadas de manera distinta, por razón de la enorme diferencia de coste entre la ejecución de instrucciones máquina y la ejecución de acciones por vehículos robots o manipuladores. Linden (1991) propuso el cruce de ideas fértiles entre ambos campos.

Retrospectivamente, es ahora posible observar cómo los mejores algoritmos de planificación clásica se han ido dirigiendo hacia versiones extendidas para dominios que contemplaban la incertidumbre. Las técnicas basadas en búsquedas se transformaron en las búsquedas en el espacio de creencias (Bonet y Geffner, 2000); SATPLAN se dirigió hacia una versión estocástica de SATPLAN (Majercik y Littman, 1999) y hacia planificación utilizando lógica booleana cuantificada (Rintanen, 1999); la planificación de orden parcial se dirigió hacia UWL (Etzioni *et al.,* 1992), CNLP (Peot y Smith, 1992), y CASSANDRA (Pryor y Collins, 1996). GRAPHPLAN se orientó hacia Sensory Graphplan o SGP (Weld *et al.,* 1998), aunque un GRAPHPLAN probabilístico ya había sido desarrollado.

El primero de los mejores sistemas de vigilancia de ejecución fue PLANEX (Fikes *et al*., 1972), que funcionaba con el planificador STRIPS para controlar el robot Shakey. PLANEX utilizó tabulación triangular (esencialmente un modo de almacenaje eficiente para las precondiciones de los planes y para cada instante de tiempo) que permitiese el reestablecimiento del plan, interrumpido por errores parciales de ejecución, sin replanificación completa. El modelo de ejecución de Shakey es discutido en el Capítulo 25. El planificador NASL (McDermott, 1978a) trataba problemas de planificación simplemente como una especificación para realizar una acción compleja, de modo que la ejecución y planificación estuviesen completamente unificadas. Se utilizaba demostración automática de teoremas para razonar acerca de estas complejas acciones.

SIPE (*System for Interactive Planning and Execution monitoring*) (Wilkins, 1988, 1990) fue el primer planificador en trabajar sistemáticamente con el problema de la replanificación. Se utilizó en proyectos de demostración de diferentes dominios, incluyendo operaciones de planificación de la plataforma aérea de un sistema de comunicación de aeronaves y en la programación de tareas para la industria cervecera australiana. Otro estudio en el que se utilizó STRIPS consistió en un plan para la construcción de edificios de varias plantas, uno de los dominios más complejos que se han abordado con un planificador.

IPEM (*Integrated Planning, Execution and Monitoring*) (Ambros-Ingerson y Steel, 1988) fue el primer sistema en integrar planificación de orden parcial y ejecución para obtener un agente de planificación continua. El AGENTE-POP-CONTINUO que hemos introducido combina ideas de IPEM, de PUCCINI (Golden, 1998), y el sistema CYPRESS (Wilkins *et al*., 1995).

A mediados de la década de los 80, se pensaba que mediante planificación de orden parcial o técnicas similares no se podría nunca generar un comportamiento de respuesta de agente suficientemente rápido para sobrevivir en el mundo real (Agre y Chapman, 1987). Por esta razón, los sistemas de **planificación reactiva** fueron propuestos; en su forma básica, éstos eran agente reflejos, con posibilidad de poseer un estado interno, que podía ser implementado con muchas de las diferentes representaciones de reglas condición-acción. La arquitectura de subsunción de Brooks (1986), que se estudia con más detalle en los Capítulos 7 y 25, utilizaba máquinas de estado-finito con varias capas, en robots con patas y ruedas, para controlar su movimiento y para que evitasen obstáculos. Pengi

PLANIFICACIÓN REACTIVA

(Agre y Chapman, 1987) fue capaz de desenvolverse eficazmente en un (completamente observable) vídeo-juego utilizando únicamente circuitos booleanos combinados con una representación «visual» de los objetivos actuales y el estado interno del agente.

Los «planes universales» (Schoppers, 1987) fueron desarrollados vía planificación reactiva como un mecanismo que mira-por-encima-datos tabulados, pero que redescubre y aplica la idea de **principios** que se habían introducido en los procesos de decisión de Markov. Un plan universal (o principio) consiste en una correspondencia entre cualquier estado y la acción que debería ser tomada en ese estado. Ginsberg (1989) rechazó efusivamente la estrategia de planes universales, porque algunas de las formulaciones de los problemas de planificación reactiva incluían resultados de intratabilidad. Schoppers (1989), por su parte, ofreció una vivaz réplica.

PRINCIPIOS

Como es frecuente, un enfoque híbrido surgió para deshacer la controversia. Utilizando jerarquías bien-diseñadas, planificadores RJT, tales como PRS (Georgeff y Lansky, 1987) y RAP (Firby, 1996), tanto como agentes de planificación continua, se pudieron alcanzar tiempos de respuesta y comportamiento complejo en planificaciones a largo plazo, y en gran variedad de dominios.

La planificación multiagente ha saltado a la popularidad recientemente, aunque en realidad tiene una larga historia. Konolige (1982) proporcionó una formalización de la planificación multiagente con lógica de primer-órden, mientras que Pednault (1986) proporcionó una descripción de tipo STRIPS. La noción de intención conjunta, esencial si los agentes tienen que ejecutar planes conjuntos, proviene del original trabajo sobre actos de comunicación (Cohen y Levesque, 1990; Cohen *et al*., 1990). El tratamiento que hemos presentado sobre planificación multicuerpo de orden parcial está basado en el trabajo de Boutilier y Brafman (2001).

Apenas se ha tratado por encima el problema de la negociación en planificación multiagente. Durfee y Lesser (1989) estudian cómo pueden ser compartidas las tareas entre agentes a través de la negociación. Kraus *et al*. (1991) describió un sistema para practicar Diplomacia, un juego complejo que requiere negociación, formación de coaliciones y disolución, y corrupción. Stone (2000) muestra cómo los agentes pueden cooperar como compañeros de equipo en entornos competitivos, dinámicos y parcialmente observables de robots futbolistas. (Weiss, 1999) es una amplia revisión de los sistemas multiagentes.

El modelo de agente-pájaro citado en el Apartado 12.7 se debe a Reynolds (1987), que ganó un óscar de la Academia por la aplicación a la simulación de bandadas de murciélagos y de pingüinos en *Batman Returns*.

EJERCICIOS

12.1 Examine cuidadosamente la representación de tiempo y recursos en la Sección 12.1.

a) ¿Por qué es una buena idea que *Duración*(*d*) sea un efecto de una acción, más que tener un campo separado de la acción y de la forma DURATION: *d*? (*Pista:* considérense efectos condicionales y disyuntivos.)

b) ¿Por qué es RECURSO*: m* un campo separado de la acción, y no un efecto?

RECURSO FUNGIBLE

12.2 Un **recurso fungible** es un recurso que es (parcialmente) utilizado por una acción. Por ejemplo, los elementos de sujeción de un coche requieren tornillos, que una vez usados no están disponibles para otros usos.

a) Explique cómo modificar la representación de la Figura 12.3 para que existan 100 tornillos inicialmente (la máquina E_1 requiere 40 unidades, y la E_2 unos 50). Las funciones símbolo + y − pueden ser usadas en literales para recursos.

b) Explique cómo la definición de **conflicto** entre enlaces causales y acciones en planificación de orden parcial debe ser modificada para tratar con recursos fungibles.

c) Algunas acciones (por ejemplo, el suministro a la industria de tornillos o de fuel en los coches) pueden *incrementar* la disponibilidad de recursos. Un recurso es monótonamente decreciente si no existe acción que lo incremente. Explique cómo usar esta propiedad para podar opciones en el espacio de búsqueda.

12.3 Proponga descomposiciones para las fases *ContratarConstructor* y *ObtenerPermiso* de la Figura 12.7, y muestre cómo los planes descompuestos se conectan entre sí en el plan global.

12.4 Proponga un ejemplo, en el dominio de la construcción de casas, de dos subplanes abstractos que no puedan emerger en un plan consistente si no se contemplan etapas compartidas. (*Pista:* sitios donde dos partes de la casa converjan son también lugares donde dos subplanes interactúan.)

12.5 Varios autores dicen que la ventaja de la planificación RJT es que puede resolver problemas del tipo «organizar un viaje desde Los Angeles a Nueva York y volver» que son difíciles de expresar en notación no-RJT porque los estados inicial y final coinciden (*En*(*LA*)). ¿Es posible encontrar una forma de representar y resolver este problema sin RJTs?

12.6 Muestre cómo una descripción de acciones STRIPS estándar puede ser descrita como una descomposición RJT, utilizando la notación *Alcanzado*(*p*) para denotar la *actividad* de alcanzar la condición *p*.

12.7 Algunas de las operaciones de los lenguajes de programación pueden ser modeladas como acciones que cambian el estado del mundo. Por ejemplo, la operación copias de contenidos asigna una localización en la memoria, mientras que la operación imprimir cambia el estado del *stream* de salida. Un programa que conste de estas operaciones puede también ser considerado un plan, cuyo objetivo viene dado por la especificación de un programa. Por tanto, los algoritmos de planificación pueden ser usados para construir programas que alcancen una especificación dada.

a) Escriba un esquema de operador para el operador asignación (hacer corresponder el valor de una variable a otra). Recuérdese que el valor original será sobrescrito.

b) Muestre cómo la creación de objetos puede ser usada por un planificador que produzca un plan para el intercambio de valores de dos variables utilizando una variable temporal.

12.8 Considérese el siguiente argumento: en un marco que permita estados iniciales con incertidumbre, los **efectos disyuntivos** son sólo una notación de conveniencia, no

una fuente de poder representacional adicional. Para cualquier esquema de acciones con efectos disyuntivos, tal como $P \vee Q$, podremos reemplazarla con efectos condicionales **cuando** R: $P \wedge$ **cuando** $\neg R$: Q, lo que a su vez puede ser reducido a dos acciones regulares. La proposición R establece una proposición aleatoria que es desconocida en el estado inicial y para la cual no existen acciones sensoriales. ¿Es éste un argumento correcto? Considere de forma separada dos casos, uno en el que una instancia del esquema de acción *a* esté en el plan, y otra en la que más de una instancia esté.

12.9 ¿Por qué la planificación condicional no puede tratar con la indeterminación ilimitada?

12.10 En el mundo de los bloques nos vimos forzados a introducir acciones STRIPS, *Mover* y *MoverseHaciaMesa,* con el fin de mantener el predicado *Limpiar* correcto. Muestre cómo los efectos condicionales pueden ser usados para representar ambos casos con una única acción.

12.11 Los efectos condicionales se presentaron para la acción *Aspirar* en el mundo aspiradora (que limpia en función de cuál sea la localización que ocupa el robot). ¿Podría pensarse en un nuevo conjunto de variables proposicionales para definir estados del mundo aspiradora, de modo que *Aspirar* tenga una descripción no-condicional? Escríbanse las descripciones *Aspirar*, *Izquierda* y *Derecha*, usando éstas nuevas proposiciones, y demuéstrese que son suficientes para describir todos los estados posibles del mundo.

12.12 Escriba la descripción completa de *Aspirar* para el problema de la aspiradora «doble Murphy» que algunas veces depositaba suciedad cuando se movía a un destino ya limpio y otras veces depositaba suciedad si *Aspirar* era aplicado a localizaciones limpias.

12.13 Encuentre una moqueta adecuadamente sucia, libre de obstáculos, y aplíquele una aspiradora. Dibuje la ruta tomada por la aspiradora tan aproximada como se pueda. Explique el porqué de esta ruta relacionándola con las formas de planificación discutidas en el capítulo.

12.14 Las siguientes citas están tomadas de la parte posterior de botes de champú. Identifique cuál de ellas es no-condicional, condicional o un plan de vigilancia de ejecución. (a) «Enjabonar. Aclarar. Repetir». (b) «Aplicar champú en el cuero cabelludo y dejarlo durante varios minutos. Aclarar y repetir si fuese necesario». (c) «Visitar un médico si el problema persiste».

12.15 El algoritmo BÚSQUEDA EN GRAFOS AND-OR de la Figura 12.10 chequea estados repetidos solamente sobre la ruta que va de la raíz al estado actual. Supongamos que, además, el algoritmo almacena datos cada instante de tiempo y los chequea con la lista (*véase* BÚSQUEDA EN GRAFOS en la Figura 3.19, para un ejemplo). Determine la información que debería ser almacenada y cómo el algoritmo debería usar esa información cuando es encontrado un estado repetido. (*Pista*: se necesitará distinguir entre estados para los cuales un subplan de éxito se construya previamente y estados a los que no se les puede encontrar ningún subplan.) Explique cómo usar **etiquetas** para evitar tener muchas copias de los subplanes.

12.16 Explique de modo preciso cómo modificar el algoritmo BÚSQUEDA EN GRAFOS AND-OR para generar un plan cíclico si no existe plan acíclico. Se necesitará tratar con tres elementos: etiquetar las etapas de los planes de modo que un plan cíclico pueda apuntar de nuevo a una parte anterior del plan, modificando la BÚSQUEDA-OR de forma que continúe buscando planes acíclicos después de encontrar un plan cíclico, y aumentar la representación del plan para indicar si un plan es cíclico. Muestre cómo el algoritmo trabaja en (a) el mundo de la aspiradora «triple Murphy», y (b) el opuesto mundo «doble Murphy». Si se puede, utilizar un computador para chequear resultados. ¿Puede ser escrito el plan para el caso (b) utilizando sintaxis de bucles estándar?

12.17 Especifique un procedimiento de actualización de estados de creencias para entornos parcialmente observables. Esto es, un método para computar la representación de un nuevo estado de creencias (como una lista de proposiciones de conocimiento) partiendo de la representación del estado actual de creencias y la representación de acciones con efectos condicionales.

12.18 Escriba descripciones de acciones, análogas a la Ecuación (12.2), para las acciones *Derecha* y *Aspirar*. De igual forma, escriba una descripción para *ChequearLocalización*, análoga a la Ecuación (12.3). Repetirlo usando el conjunto de proposiciones alternativo del Ejercicio 12.11.

12.19 Eche un vistazo a la lista del Apartado 12.6 acerca de las cosas que el agente de replanificación no puede hacer. Esboce un algoritmo que pueda tratar una o más de éstas limitaciones.

12.20 Considérese el siguiente problema: un paciente llega a la consulta del médico con síntomas que pueden haber sido provocados por deshidratación o enfermedad *D* (pero no por ambas). Existen dos posibles acciones: *Beber*, lo que cura la deshidratación, y *Medicar* que cura la enfermedad *D*, pero que tiene un efecto colateral poco deseable si el paciente lo toma cuando está deshidratado. Escriba la descripción del problema en PDDL, y haga un diagrama del plan sin sensores que resuelva el problema, enumerando todos los mundos posibles que sean relevantes.

12.21 Para el problema de la medicación del ejercicio anterior, añada una acción *Test* que tenga el efecto condicional *EnriquecimientoCultural* cuando la *Enfermedad* es verdad y que para cualquier caso tenga el efecto perceptivo *Conocido*(*EnriquecimientoCultural*). Dibuje un diagrama de un plan condicional que resuelva el problema y minimice el uso de la acción *Medicar.*

13 Incertidumbre

Donde se verá lo que un agente debería hacer cuando no todo está claro cristalino.

13.1 Comportamiento bajo incertidumbre

INCERTIDUMBRE

Los agentes lógicos que se describieron en las Partes III y IV hacen las asignaciones epistemológicas de que las proposiciones son ciertas, falsas o desconocidas. Cuando un agente conoce hechos suficientes sobre su entorno, el enfoque de la lógica le permite obtener planes con garantías para su desarrollo. Esto está bien. Desafortunadamente, *los agentes casi nunca tienen acceso a toda la verdad sobre su entorno*. Los agentes deben, así, saber comportarse bajo **incertidumbre**. Por ejemplo, un agente en el mundo *wumpus* del Capítulo 7 tiene sensores que le muestran sólo información local; la mayor parte del mundo no se observa inmediatamente. Un agente *wumpus* se encontrará a menudo incapaz de descubrir cuál de los dos cuadrados contiene un pozo. Si esos cuadrados están en la ruta hacia el oro, entonces el agente tendría que arriesgarse y entrar en uno de los dos cuadrados.

El mundo real es mucho más complejo que el mundo *wumpus*. Para un agente lógico, sería imposible construir una descripción completa y exacta de cómo se desarrollarán sus acciones. Suponga, por ejemplo, que el agente quiere conducir algo para tomar un vuelo y considera un plan, A_{90}, que supone dejar la casa 90 minutos antes de que salga el vuelo y conducir a una velocidad razonable. Incluso aunque el aeropuerto esté tan sólo a unas 15 millas, el agente no inferirá con certidumbre que «El Plan A_{90} nos llevará al aeropuerto a tiempo». En lugar de eso, llega a la conclusión más débil «El Plan A_{90} nos llevará al aeropuerto a tiempo, siempre que mi coche no se averíe o se quede sin gasolina, y no tenga un accidente, y no haya accidentes en el puente, y el avión

no salga anticipadamente, y…». Ninguna de estas condiciones pueden deducirse, por lo que no puede inferirse el éxito del plan. Este es un ejemplo del **problema de la cualificación** mencionado en el Capítulo 10.

Si un agente lógico no puede concluir que cualquier secuencia concreta de acciones alcance su objetivo, entonces será incapaz de actuar. La planificación condicional puede derrotar a la incertidumbre hasta cierto punto, sólo si las acciones del agente pueden obtener la información que se requiere y sólo si no hay demasiados imprevistos diferentes. Otra posible solución sería dotar al agente con una simple pero errónea teoría del mundo que le *permita* deducir un plan; presumiblemente, esos planes funcionarán *la mayoría* de las veces, pero los problemas se presentan cuando los acontecimientos contradicen la teoría del agente. Más aún, el tratamiento del equilibrio entre la precisión y la utilidad de la teoría del agente parece en sí mismo que requiere razonamiento sobre incertidumbre. En resumen, los agentes que no sean puramente lógicos serán capaces de concluir que el plan A_{90} es el correcto a desempeñar.

Aún con todo, supongamos que A_{90} es el apropiado a realizar. ¿Qué queremos decir al afirmar esto? Como discutimos en el Capítulo 2, queremos decir que de todos los planes que podrían llevarse a cabo, se espera que el A_{90} maximice la medida de desarrollo del agente, dada la información que tiene sobre el entorno. La medida de desarrollo incluye el llegar a tiempo al aeropuerto para el vuelo, evitar una espera larga e improductiva en el aeropuerto, y evitar prisas en poner etiquetas cerca del pasaje. La información que tiene el agente no puede garantizar ninguno de estos resultados finales para A_{90}, pero puede proporcionar algún grado de creencia de que pueda lograrse. Otros planes, tal como el A_{120}, podrían incrementar la creencia del agente de que llegará al aeropuerto a tiempo, pero también pueden incrementar la posibilidad de una mayor espera. *Lo correcto a realizar (la* ***decisión racional****) depende tanto de la importancia relativa de los distintos objetivos como de la verosimilitud y el grado con el cual se conseguirán*. El resto de esta sección afina estas ideas, como parte de los preparativos del desarrollo de las teorías generales de razonamiento incierto y decisiones racionales que presentamos en este y subsiguientes capítulos.

Manipulación del conocimiento incierto

En esta sección veremos más detenidamente la naturaleza del conocimiento incierto. Usaremos un ejemplo de diagnóstico sencillo para ilustrar los conceptos involucrados. El diagnóstico (ya sea para medicina, reparación de automóviles, o lo que quiera) es una tarea que casi siempre involucra incertidumbre. Intentemos escribir las reglas para diagnóstico dental usando la lógica de predicados de primer orden, para que así podamos ver cómo fracasa la aproximación lógica. Considere la siguiente regla:

$$\forall p\ Síntoma(p, Dolor\text{-}de\text{-}muelas) \Rightarrow Enfermedad(p, Caries)$$

El problema es que esta regla es errónea. No todos los pacientes con dolor de muelas tienen caries; algunos tienen dolencia de encías, una acumulación de pus (absceso), u otro problema distinto:

$$\forall p\ Síntoma(p, Dolor\text{-}de\text{-}muelas) \Rightarrow Enfermedad(p,Caries) \vee Enfermedad(p,Dolor\text{-}de\text{-}muelas) \vee Enfermedad(p,Absceso)\ldots$$

Desafortunadamente, para hacer la regla cierta, tenemos que añadir una lista casi ilimitada de causas posibles. Podríamos intentar convertir la regla en una regla causal:

$$\forall p\ Enfermedad(p, Caries) \Rightarrow Sintoma(p, Dolor\text{-}de\text{-}muelas)$$

Pero esta regla no es cierta tampoco: no todas las caries causan dolencias. El único modo para arreglar la regla es hacerla lógicamente exhaustiva: incrementar el lado izquierdo de la regla con todos los requisitos necesarios para que una caries cause un dolor de muelas. Incluso entonces, para los propósitos del diagnóstico, uno debe también tener en cuenta la posibilidad de que el paciente podría tener un dolor de muelas y una caries que no estén relacionados.

Intentar usar la lógica de predicados de primer orden para hacer frente a un dominio como el diagnóstico médico de este modo falla por tres razones principales:

PEREZA

IGNORANCIA TEÓRICA

IGNORANCIA PRÁCTICA

- **Pereza:** poner en una lista el conjunto completo de antecedentes y consecuentes que se necesitan para asegurar una regla sin excepciones tiene demasiado trabajo y usar tales reglas es bastante difícil.
- **Ignorancia Teórica:** la ciencia médica no tiene una teoría completa para el dominio.
- **Ignorancia Práctica:** incluso si conocemos todas las reglas, pudiera ser incierto sobre un paciente particular, ya sea porque no se hayan realizado todos los chequeos necesarios o porque no puedan realizarse.

La conexión entre dolor de muelas y caries no es exactamente una consecuencia lógica en ninguna dirección. Esto es típico en el dominio médico, al igual que en la mayoría de otros dominios sentenciosos: leyes, negocios, diseño, reparación de automóviles, jardinería, casamientos, etc. El conocimiento del agente puede como mucho proporcionar sólo un **grado de creencia** en las oraciones relevantes. Nuestra principal herramienta para tratar con grados de creencia será la **teoría de la probabilidad**, que asigna a cada oración un grado numérico de creencia entre 0 y 1. (Algunos métodos alternativos para razonamiento incierto se tratan en la Sección 14.7.)

GRADO DE CREENCIA

TEORÍA DE LA PROBABILIDAD

La probabilidad proporciona una manera de ***resumir*** *la incertidumbre que se deriva de nuestra pereza e ignorancia*. No podríamos saber con seguridad lo que aqueja a un paciente particular, pero podríamos creer que hay, digamos, un 80 por ciento de posibilidad (esto es, una probabilidad de 0,8) de que el paciente tiene una caries si él o ella tiene un dolor de muelas. Esto es, hasta donde llega el conocimiento del agente esperamos que de todas las situaciones que son indistinguibles de la situación actual, el paciente tendrá una caries en el 80 por ciento de ellas. Esta creencia podría provenir de datos estadísticos (el 80 por ciento de los pacientes observados hasta este momento con dolor de muelas han tenido caries) o de algunas reglas generales, o de una combinación de fuentes de indicios. El 80 por ciento resume aquellos casos en los que están presentes todos los factores necesarios para una caries que causa un dolor de muelas y otros casos en los que el paciente tiene tanto dolor de muelas como caries pero los dos están desconectados. El 20 por ciento restante resume todas las otras causas posibles de dolor de muelas para los que somos demasiado perezosos o ignorantes en confirmar o desmentir.

Asignar probabilidad 0 a una oración determinada corresponde a una creencia inequívoca de que la oración es falsa, mientras que asignar una probabilidad de 1 corres-

ponde a una creencia rotunda de que la oración es cierta. Las probabilidades entre 0 y 1 corresponden a grados intermedios de creencia en la veracidad de la oración. La oración en sí misma es *de hecho* o verdadera o falsa. Es importante notar que el grado de creencia es diferente de un grado de veracidad. Una probabilidad de 0,8 no significa «80 por ciento verdadero» sino un 80 por ciento de grado de creencia, eso es, una expectativa realmente fuerte. Así, la teoría de la probabilidad cumple la misma obligación ontológica que la lógica (a saber, los hechos o se sostienen o no en el mundo). Los grados de veracidad, a diferencia de los grados de creencia, es la materia de la **lógica borrosa**, que será tratada en la Sección 14.7.

En lógica, una oración tal como «El paciente tiene una caries» es verdadera o falsa dependiendo de la interpretación del mundo; es verdad sólo cuando el hecho al que hace referencia es el caso. En la teoría de la probabilidad, una oración tal como «La probabilidad de que el paciente tiene una caries es 0,8» hace referencia a las creencias del agente, no directamente sobre el mundo. Estas creencias dependen de las percepciones que el agente ha recibido hasta el momento. Estas percepciones constituyen la **evidencia** sobre la que se basan las afirmaciones de probabilidades. Por ejemplo, suponga que el agente saca una carta de un mazo ya barajado. Antes de mirar la carta, el agente asignaría una probabilidad de $1/52$ de que se trata del as de picas. Después de mirar la carta, una probabilidad adecuada para la misma proposición debería estar entre 0 y 1. Así, una asignación de probabilidad a una proposición es análogo a decir si una oración lógica determinada (o su negación) está producida por la base de conocimiento, más que si es o no cierta. Al igual que el estatus de lo producido puede cambiar cuando se añaden más oraciones a la base de conocimiento, las probabilidades pueden cambiar cuando se adquiere más evidencia[1].

EVIDENCIA

Todas las oraciones de probabilidad deben así indicar la evidencia con respecto a la cual se está calculando la probabilidad. Cuando un agente recibe nuevas percepciones, sus valoraciones de probabilidad se actualizan para reflejar la nueva evidencia. Antes de que la evidencia se obtenga, hablaremos de probabilidad *a priori* o **incondicional**; después de obtener la evidencia, hablaremos de probabilidad *a posteriori* o **condicional**. En la mayoría de los casos, un agente tendrá alguna evidencia de sus percepciones y será interesante calcular las probabilidades posteriores de las consecuencias que le preocupan.

Incertidumbre y decisiones racionales

La presencia de incertidumbre cambia radicalmente el modo en que un agente toma decisiones. Un agente lógico tiene típicamente un objetivo y ejecuta cualquier plan que le garantice alcanzarlo. Una acción puede seleccionarse o rechazarse sobre la base de si alcanza el objetivo, sin tener en cuenta lo que otras acciones podrían alcanzar. Pero este no es el caso cuando la incertidumbre interviene en la situación. Consideremos de nue-

[1] Esto es completamente diferente a como llegue a ser la verdad o falsedad de una oración cuando el mundo cambia. Manejar mundos cambiantes vía probabilidades requiere los mismos tipos de mecanismos (situaciones, intervalos y eventos) que los usados en el Capítulo 10 para representaciones lógicas. Estos mecanismos se discuten en el Capítulo 15.

vo el plan A_{90} para llegar al aeropuerto. Suponga que tiene un 95 por ciento de posibilidad de que surta efecto. ¿Esto significa que es una elección racional? No necesariamente: puede que estén otros planes, como el A_{120}, con probabilidades más altas de que den resultado. Si es vital o no perder el vuelo, entonces vale arriesgarse a una mayor espera en el aeropuerto. ¿Y qué tal el A_{1440}, un plan que involucra dejar la casa en más de 24 horas? En la mayoría de las circunstancias, esto no es una buena elección, porque, aunque casi garantiza llegar a tiempo, involucra una espera intolerable.

PREFERENCIAS

CONSECUENCIAS

TEORÍA DE LA UTILIDAD

Para tomar tales elecciones, un agente debe primero tener **preferencias** entre las diferentes **consecuencias** posibles de los diversos planes. Una consecuencia particular es un estado completamente especificado, incluyendo factores tales como si el agente llega a tiempo y el tiempo de espera en el aeropuerto. Usaremos la **teoría de la utilidad** para representar y razonar con preferencias. (El término **utilidad** se usa aquí en el sentido de «la calidad de ser útil», no en el sentido de compañía eléctrica o planta purificadora de agua.) La teoría de la utilidad establece que cada estado tiene un grado de utilidad, o beneficio, para un agente y que el agente preferirá los estados con mayor utilidad.

La utilidad de un estado concierne al agente cuyas preferencias supuestamente están representadas por la función de utilidad. Por ejemplo, las funciones de pago para juegos en el Capítulo 6 son funciones de utilidad. La utilidad de un estado en el que las blancas han ganado un juego de ajedrez es obviamente elevada para el agente que juega blancas, pero baja para el agente que juega negras. De nuevo, algunos jugadores (incluyendo los autores) serían felices con unas tablas frente al campeón del mundo, mientras que otros jugadores (incluyendo el antiguo campeón del mundo) podrían no serlo. Sobre gustos o preferencias no hay nada escrito: podría pensar que un agente que prefiere helado de pompas de chicle de jalapeño a patatas fritas de chocolate es extraño o incluso equivocado, pero no podría decir que el agente es irracional. Una función de utilidad puede incluso explicar el comportamiento altruista, simplemente incluyendo el «bienestar» de otros como uno de los factores que contribuyen a la utilidad propia del agente.

TEORÍA DE LA DECISIÓN

Las preferencias, expresadas como utilidades, se combinan con probabilidades en la teoría general de decisiones racionales llamada **teoría de la decisión**:

Teoría de la Decisión = teoría de la probabilidad + teoría de la utilidad

La idea fundamental de la teoría de la decisión es que *un agente es racional si y sólo si elige la acción que le produce la utilidad esperada más alta, en promedio sobre todas las posibles consecuencias de la acción.* Esto es conocido como el principio de **Máxima Utilidad Esperada** (MUE). Vimos este principio en acción en el Capítulo 6 cuando tocamos brevemente las decisiones óptimas para el *backgammon*. Veremos que es de hecho un principio completamente general.

Diseño de un agente de decisión teórico

La Figura 13.1 esboza la estructura de un agente que usa la teoría de la decisión para seleccionar acciones. El agente es idéntico, a un nivel abstracto, al agente lógico descrito en el Capítulo 7. La diferencia principal es que el conocimiento del agente de de-

ESTADO DE CREENCIA

cisión teórico para el estado en curso es incierto; el **estado de creencia** del agente es una representación de las probabilidades de todos los posibles estados actuales del mundo. Cuando pasa el tiempo, el agente acumula más evidencia y su estado de creencia cambia. Dado el estado de creencia, el agente puede hacer predicciones probabilísticas de las consecuencias de la acción y así seleccionar la acción con utilidad esperada más alta. Este capítulo y el siguiente se centran en la tarea de la representación y computación con información probabilista en general. El Capítulo 15 está relacionado con los métodos para las tareas específicas de la representación y la actualización del estado de creencia y predicción del entorno. El Capítulo 16 recorre la teoría de la utilidad en mayor profundidad, y el Capítulo 17 desarrolla algoritmos para tomar decisiones complejas.

```
función AGENTE-DT (percepción) retorna una acción
  estático: estado_de_creencia, creencias probabilistas sobre el estado actual del mundo
            accion, la acción del agente

  actualizar estado_de_creencia basado en acción y percepción
  calcular las probabilidades de las consecuencias para las acciones,
    dadas las descripciones de las acciones y el actual estado_de_creencia
  seleccionar acción con utilidad esperada más alta,
    dadas las probabilidades de las consecuencias y la información de utilidad
  retorna acción
```

Figura 13.1 Un agente de decisión teórico que selecciona acciones racionales. Los pasos serán desarrollados en los cinco capítulos siguientes.

13.2 Notación básica con probabilidades

Ahora que hemos establecido el marco general para un agente racional, necesitaremos un lenguaje formal para la representación y el razonamiento con conocimiento incierto. Cualquier notación para describir grados de creencia debe ser capaz de tratar dos cuestiones principales: la naturaleza de las oraciones para las que se asignan grados de creencia y la dependencia del grado de creencia en la experiencia del agente. La versión de la teoría de la probabilidad que presentamos usa una extensión de la lógica proposicional para sus oraciones. La dependencia en la experiencia se refleja en la distinción sintáctica entre la probabilidad priori de las oraciones, que se aplica antes de que se obtenga cualquier evidencia, y la probabilidad condicional de las oraciones, que incluye la evidencia explícitamente.

Proposiciones

Los grados de creencia se aplican siempre a las **proposiciones** (afirmaciones de que tal o cual es el caso). Hasta ahora hemos visto dos leguajes formales (la lógica proposicio-

nal y la lógica de primer orden) para la declaración de proposiciones. La teoría de la probabilidad típicamente usa un lenguaje que es ligeramente más expresivo que la lógica proposicional. Esta sección describe ese lenguaje. (La Sección 14.6 discute modos para atribuir grados de creencia a afirmaciones en la lógica de primer orden.)

VARIABLE ALEATORIA

El elemento básico del lenguaje es la **variable aleatoria**, que puede pensarse como algo que se refiere a una «parte» del mundo cuyo estatus es desconocido inicialmente. Por ejemplo, *Caries* podría referirse a si mi muela del juicio inferior izquierda tiene una caries. Las variables aleatorias representan un papel similar al de las variables PSR en problemas de satisfacción de restricciones y al de los símbolos proposicionales en lógica proposicional. Siempre escribiremos en mayúsculas los nombres de las variables aleatorias (sin embargo, también usaremos nombres en una letra minúscula para representar una variable aleatoria desconocida, por ejemplo: $P(a) = 1 - P(\neg a)$).

DOMINIO

Cada variable aleatoria tiene un **dominio** de posibles valores que puede tomar. Por ejemplo, el dominio de *Caries* podría ser $\langle cierto, falso \rangle$[2]. (Usaremos minúsculas para los nombres de los valores.) El tipo más sencillo de proposición afirma que una variable aleatoria tiene un valor particular tomado de su dominio. Por ejemplo $Caries = cierto$ representaría la proposición de que en efecto tengo una caries en mi muela del juicio inferior izquierdo.

Como con las variables PSR, las variables aleatorias están típicamente divididas en tres clases, dependiendo del tipo de dominio:

VARIABLES ALEATORIAS BOOLEANAS

- **Las variables aleatorias booleanas**, tal como *Caries*, tienen el dominio $\langle cierto, falso \rangle$. A menudo abreviaremos una proposición como $Caries = cierto$ simplemente por el nombre en minúsculas *caries*. Igualmente, $Caries = falso$ se abreviaría por $\neg caries$.

VARIABLES ALEATORIAS DISCRETAS

- **Las variables aleatorias discretas**, que incluyen las variables aleatorias booleanas como un caso especial, toman valores en un dominio contable. Por ejemplo, el dominio de *Tiempo* sería $\langle soleado, lluvioso, nublado, nevado \rangle$. Los valores del dominio deben ser mutuamente exclusivos y exhaustivos. Cuando no haya confusión, usaremos, por ejemplo, *nevado* como una abreviación de $Tiempo = nevado$.

VARIABLES ALEATORIAS CONTINUAS

- **Las variables aleatorias continuas** toman sus valores de los números reales. El dominio puede ser o toda la recta real o algún subconjunto como el intervalo [0,1]. Por ejemplo, la proposición $X = 4{,}02$ afirma que la variable aleatoria X toma el valor exacto 4,02. Las proposiciones concernientes a las variables aleatorias continuas pueden ser también desigualdades, como $X \leq 4{,}02$.

Con algunas excepciones, centraremos nuestra atención en el caso discreto.

Básicamente, las proposiciones como $Caries = cierto$ y $Dolor\text{-}de\text{-}muelas = falso$, pueden combinarse para construir proposiciones complejas usando todos los conectivos lógicos estándares. Por ejemplo, $Caries = cierto \wedge Dolor\text{-}de\text{-}muelas = falso$ es una proposición a la que uno puede asignar un grado de creencia o escepticismo. Como se explicó en el párrafo previo, esta proposición puede también escribirse como $caries \wedge \neg dolor\text{-}de\text{-}muelas$.

[2] Quizás esperara que el dominio se escribiera como un conjunto: $\{cierto, falso\}$. Lo escribimos como una tupla porque posteriormente nos convendrá para imponer una ordenación en los valores.

Sucesos atómicos

SUCESO ATÓMICO

La noción de **suceso atómico** es útil para entender los fundamentos de la teoría de la probabilidad. Un suceso atómico es una especificación *completa* del estado del mundo sobre el que el agente está inseguro. Puede pensarse en él como la asignación de valores particulares de todas las variables que componen el mundo. Por ejemplo, si mi mundo consta sólo de las variables booleanas *Caries* y *Dolor-de-muelas*, entonces hay exactamente cuatro sucesos atómicos; la proposición $Caries = falso \wedge Dolor\text{-}de\text{-}muelas = cierto$ es uno de tales sucesos[3].

Los sucesos atómicos tienen algunas propiedades importantes:

- Son mutuamente exclusivos, a lo sumo sólo uno puede darse. Por ejemplo, $caries \wedge dolor\text{-}de\text{-}muelas$ y $caries \wedge \neg dolor\text{-}de\text{-}muelas$ no pueden darse simultáneamente.
- El conjunto de todos los sucesos atómicos es exhaustivo, al menos debe darse uno. Es decir, la disyunción de todos los sucesos atómicos es equivalente lógicamente a *cierto*.
- Cualquier suceso atómico implica la veracidad o falsedad de cada proposición, ya sea simple o compleja. Esto puede verse usando la semántica estándar de los conectivos lógicos (Capítulo 7). Por ejemplo, el suceso atómico $caries \wedge \neg dolor\text{-}de\text{-}muelas$ conlleva la verdad de *caries* y la falsedad de $caries \Rightarrow dolor\text{-}de\text{-}muelas$.
- Cualquier proposición es lógicamente equivalente a la disyunción de todos los sucesos atómicos que implica la verdad de la proposición. Por ejemplo, la proposición *caries* es equivalente a la disyunción de los sucesos atómicos $caries \wedge dolor\text{-}de\text{-}muelas$ y $caries \wedge \neg dolor\text{-}de\text{-}muelas$.

El Ejercicio 13.4 le pide que pruebe algunas de estas propiedades.

Probabilidad priori

INCONDICIONAL

PROBABILIDAD PRIORI

La **probabilidad *a priori*** o **incondicional** asociada a una proposición a es el grado de creencia que se le otorga *en ausencia de cualquier otra información*; y se escribe como $P(a)$. Por ejemplo, si la probabilidad priori de que tenga una caries es 0,1, entonces deberíamos escribir

$$P(Caries = cierto) = 0{,}1 \quad \text{o} \quad P(caries) = 0{,}1.$$

Es importante recordar que $P(a)$ puede usarse sólo cuando no hay otra información. Tan pronto como se conozca nueva información, debemos razonar con la probabilidad *condicional* de a dada esa nueva información. Las probabilidades condicionales se tratarán en la sección siguiente.

En ocasiones, querremos hablar de las probabilidades de todos los valores posibles de una variable aleatoria. En ese caso, usaremos una expresión como $\mathbf{P}(Tiempo)$, que de-

[3] Muchas formulaciones estándares de la teoría de la probabilidad consideran los sucesos atómicos, también conocidos por **puntos maestrales**, como proposiciones primitivas y definen una variable aleatoria como una función que toma como entrada a un suceso atómico y devuelve un valor del dominio apropiado. Esta aproximación es quizá más general, pero también es menos intuitiva.

nota un *vector* de valores que corresponden a las probabilidades de cada estado individual del tiempo. Así, en vez de escribir las cuatro ecuaciones

$$P(Tiempo = soleado) = 0{,}7$$
$$P(Tiempo = lluvioso) = 0{,}2$$
$$P(Tiempo = nuboso) = 0{,}08$$
$$P(Tiempo = nevado) = 0{,}02.$$

simplemente escribiremos

$$\mathbf{P}(Tiempo) = \langle 0{,}7, 0{,}2, 0{,}08, 0{,}02 \rangle.$$

DISTRIBUCIÓN DE PROBABILIDAD

Esta expresión define una **distribución de probabilidad** *a priori* para la variable aleatoria *Tiempo*.

También usaremos expresiones como $\mathbf{P}(Tiempo, Caries)$ para denotar las probabilidades de todas las combinaciones de los valores de un conjunto de variables aleatorias[4]. En ese caso, $\mathbf{P}(Tiempo, Caries)$ puede representarse por una tabla de probabilidades de dimensión 4×2. Ésta se llama la **distribución de probabilidad conjunta** de *Tiempo* y *Caries*.

DISTRIBUCIÓN DE PROBABILIDAD CONJUNTA

A veces será útil pensar sobre el conjunto completo de variables aleatorias que se utilicen para describir el mundo. Una distribución de probabilidad conjunta que considere este conjunto completo se llama la **distribución de probabilidad conjunta completa**. Por ejemplo, si el mundo consta exactamente de las variables *Caries*, *Dolor-de-muelas* y *Tiempo*, entonces la distribución conjunta completa viene dada por.

DISTRIBUCIÓN DE PROBABILIDAD CONJUNTA COMPLETA

$$\mathbf{P}(Caries, Dolor\text{-}de\text{-}muelas, Tiempo)$$

Esta distribución conjunta puede representarse como una tabla de dimensión $2 \times 2 \times 4$ con 16 entradas. Una distribución conjunta completa especifica la probabilidad de cada suceso atómico y es así una especificación completa de la incertidumbre que tiene uno sobre el mundo en cuestión. Veremos en la Sección 13.4 que cualquier pregunta probabilista puede responderse a partir de la distribución conjunta completa.

Para variables continuas, no es posible escribir la distribución completa como una tabla, ya que hay infinitos valores. Es más, usualmente uno define la probabilidad de que una variable aleatoria toma algún valor x como una función parametrizada de x. Por ejemplo, consideremos que la variable aleatoria X denota la temperatura máxima de mañana en Berkeley. Entonces la afirmación

$$P(X = x) = U[18, 26](x)$$

expresa la creencia de que X se distribuye uniformemente entre 18 y 26 grados Celsius. (Algunas distribuciones continuas útiles se definen en el Apéndice A.) Las distribuciones de probabilidad para variables continuas se llaman **funciones de densidad de probabilidad**. Las funciones de densidad difieren de las distribuciones discretas en su significado. Por ejemplo usando la distribución de la temperatura dada anteriormente, comprobamos que $P(X = 20{,}5) = U[18, 26](20{,}5) = 0{,}125/C$. Esto *no* significa que haya

FUNCIONES DE DENSIDAD DE PROBABILIDAD

[4] La regla general para la notación es que la distribución abarca todos los valores de las variables que están en mayúsculas. Así, una expresión como $\mathbf{P}(Tiempo, Caries)$ es un vector de probabilidades de cuatro elementos para la conjunción de cada tipo de tiempo con el caso $Caries = cierto$.

un riesgo del 12,5 por ciento de que mañana la temperatura máxima será exactamente de 20,5 grados; la probabilidad de que esto ocurra es por supuesto de cero. El significado técnico es que la probabilidad de que la temperatura está en una región pequeña alrededor de los 20,5 grados es igual, en el límite, a 0,125 dividido por el ancho de la región en grados Celsius:

$$\lim_{dx \to 0} P(20{,}5 \leq X \leq 20{,}5 + dx)/dx = 0{,}125/C$$

Algunos autores usan símbolos diferentes para distribuciones discretas y funciones de densidad; nosotros usaremos P en ambos casos, ya que en contadas ocasiones aparece confusión y las ecuaciones son usualmente idénticas. Nótese que las probabilidades son números sin unidad, mientras que las funciones de densidad están medidas con una unidad, en este caso el valor recíproco de los grados.

Probabilidad condicional

PROBABILIDAD CONDICIONAL

PROBABILIDAD *A POSTERIORI*

Una vez que el agente obtiene alguna evidencia referente a las variables aleatorias desconocidas que constituyen el dominio, las probabilidades priori ya no son aplicables. En vez de eso, usamos **probabilidades *a posteriori*** o **condicionales**. La notación que se usa es $P(a|b)$, donde a y b son proposiciones cualesquiera[5]. Esto se lee como «la probabilidad de a, supuesto que *todo lo que conozco* es b». Por ejemplo,

$$P(caries|dolor\text{-}de\text{-}muelas) = 0{,}8$$

indica que si se observa a un paciente que tiene un dolor de muelas y todavía no se tiene otra información disponible, entonces la probabilidad de que el paciente tenga una caries será 0,8. Una probabilidad *a priori*, tal como $P(caries)$, puede contemplarse como un caso especial de la probabilidad condicional $P(caries|\ )$, donde la probabilidad se condiciona a ninguna evidencia.

Las probabilidades condicionales pueden definirse en términos de probabilidades no condicionales. La ecuación que la define es:

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \tag{13.1}$$

siempre que $P(b) > 0$. Esta ecuación puede escribirse también como

$$P(a \wedge b) = P(a|b)P(b)$$

REGLA DEL PRODUCTO

que es conocida como la **regla del producto**. La regla del producto es quizá más fácil de recordar así: la regla proviene del hecho de que si a y b es cierto, necesitamos que b sea cierto, y también necesitamos que a sea cierto dado b. Podemos obtener la otra regla dándole la vuelta:

$$P(a \wedge b) = P(b|a)P(a)$$

En algunos casos, es más fácil razonar en términos de probabilidades *a priori* de conjunciones, pero mayoritariamente usaremos probabilidades condicionales como nuestro instrumento para inferencia probabilista.

[5] El operador «|» tiene la menor precedencia posible, así $P(a \wedge b|c \vee d)$ significa $P((a \wedge b)|(c \vee d))$.

Podemos también usar la notación **P** para distribuciones condicionales. $\mathbf{P}(X|Y)$ da los valores de $\mathbf{P}(X = x_i|Y = y_j)$ para cada posible i, j. Como ejemplo de cómo esto hace nuestra notación más breve, consideremos la aplicación de la regla del producto a cada caso donde las proposiciones a y b asignan valores particulares a X e Y respectivamente. Obtenemos las siguientes ecuaciones:

$$\begin{aligned}
&P(X = x_1 \wedge Y = y_1) = P(X = x_1|Y = y_1)P(Y = y_1)\\
&P(X = x_1 \wedge Y = y_2) = P(X = x_1|Y = y_2)P(Y = y_2)\\
&\vdots
\end{aligned}$$

Podemos combinarlas todas en la ecuación sencilla

$$\mathbf{P}(X, Y) = \mathbf{P}(X|Y)\mathbf{P}(Y)$$

Recordar que esto denota a un conjunto de ecuaciones relacionando las correspondientes entradas individuales en las tablas, y *no* una multiplicación matricial de las tablas.

Es tentador, pero erróneo, ver las probabilidades condicionales como si fueran implicaciones lógicas a las que se les añade incertidumbre. Por ejemplo, la oración $P(a|b) = 0{,}8$ *no puede* interpretarse para dar a entender que «si b se cumple, inferir que $P(a)$ es 0,8». Tal interpretación sería errónea por dos motivos: primero, $P(a)$ siempre denota a la probabilidad *a priori* de a, no la probabilidad *a posteriori* dada alguna evidencia; segundo, la oración $P(a|b) = 0{,}8$ es válida justo en el momento en el que b es la única evidencia disponible. Cuando se dispone de la información adicional c, el grado de creencia en a es $P(a|b \wedge c)$, que puede tener poca relación con $P(a|b)$. Por ejemplo, c pudiera decirnos directamente si a es cierto o falso. Si reconocemos a un paciente que se queja de dolor de muelas y descubrimos una caries, entonces tenemos la evidencia adicional *caries*, y concluimos (trivialmente) que $P(caries|dolor\text{-}de\text{-}muelas \wedge caries) = 1{,}0$.

13.3 Los axiomas de la probabilidad

Hasta el momento, hemos definido una sintaxis para las proposiciones y para las oraciones probabilistas prioris y condicionales sobre esas proposiciones. Ahora debemos proporcionar algún tipo de semántica para las oraciones probabilistas. Comencemos con los axiomas básicos que sirven para definir la escala probabilista y sus puntos extremos:

1. Todas las probabilidades están entre 0 y 1. Para cualquier proposición a,

$$0 \leq P(a) \leq 1$$

2. Las proposiciones necesariamente ciertas (es decir, válidas) tienen probabilidad 1, y las proposiciones necesariamente falsas (es decir, insatisfactibles) tienen probabilidad 0.

$$P(cierto) = 1 \qquad P(falso) = 0$$

¿De dónde vienen las probabilidades?

Se ha tenido un debate sin fin sobre la procedencia y estatus de los números probabilistas. La posición **frecuencialista** es que los números pueden provenir sólo de *experimentos*: si chequeamos a 100 personas y encontramos que 10 de ellas tienen una caries, entonces podemos decir que la probabilidad de una caries es aproximadamente 0,1. Desde esta perspectiva la afirmación «la probabilidad de una caries es 0,1» significa que 0,1 es la fracción que debería observarse en el límite de infinitas muestras. A partir de cualquier muestra finita, podemos estimar la fracción auténtica y también calcular cómo de exacta es nuestra estimación para que sea la más verosímil.

La perspectiva **objetiva** es que las probabilidades son aspectos reales del universo (las predisposiciones de los objetos a comportarse de cierta manera) más que la descripción exacta de un grado de creencia del observador. Por ejemplo, que en una moneda no trucada se presenten las caras con probabilidad 0,5 es una predisposición propia de la moneda. Desde esta perspectiva, las medidas frecuencialistas son intentos para advertir estas predisposiciones. La mayoría de los físicos están de acuerdo con que los fenómenos cuánticos son objetivamente probabilistas, pero la incertidumbre a escala macroscópica (por ejemplo, en lanzamientos de monedas) usualmente se origina de la ignorancia de las condiciones iniciales y no parece consistente con la perspectiva de las predisposiciones.

La perspectiva **subjetiva** describe las probabilidades como un modo de caracterizar las creencias de un agente, más que como la consideración de cualquier trasfondo físico del exterior. Esto permite al doctor o analista obtener los números, para decir, «En mi opinión supongo que la probabilidad de una caries está alrededor del 0,1». Distintas técnicas más fidedignas, como los sistemas de apuestas descritos anteriormente, también se han desarrollado para producir como respuesta estimaciones probabilistas a partir de los seres humanos.

En el fondo, incluso una posición frecuencialista estricta involucra análisis subjetivo, por lo que probablemente la diferencia tenga poca importancia práctica. El problema de la **clase de referencia** ilustra la intrusión de la subjetividad. Suponga que un doctor frecuencialista quiere conocer los riesgos de que un paciente tenga una enfermedad concreta. El doctor quiere tener en cuenta a otros pacientes que son similares en aspectos importantes (la edad, los síntomas, quizás el sexo) y ve a partir de ellos qué proporción tienen en la enfermedad. Pero si el doctor consideró todo lo conocido sobre el paciente (peso hasta el gramo más cercano, color del pelo, nombre de soltera de la madre, etc.) el resultado sería que no hay otros pacientes que sean exactamente iguales y así no hay clase de referencia de la que recoger datos experimentales. Esto ha sido un problema fastidioso en la filosofía de la ciencia.

El **principio de indiferencia** de Laplace (1816) establece que a las proposiciones que son sintácticamente «simétricas» con respecto a la evidencia se les deberían asignar la misma probabilidad. Se han propuesto diversas correcciones, culminando en el intento de Carnap y otros de desarrollar una **lógica inductiva** rigurosa, capaz de calcular la probabilidad correcta para cualquier proposición a partir de cualquier recopilación de observaciones. Actualmente, se cree que no existe una única lógica inductiva; o mejor dicho, cualquiera de esas lógicas se apoya en una distribución de probabilidad priori subjetiva cuyo efecto disminuye cuantas más observaciones se recopilen.

A continuación, necesitamos un axioma que relacione las probabilidades de las proposiciones que están ligadas en términos lógicos. El modo más sencillo para hacer esto es definir la probabilidad de una disyunción como sigue:

3. La probabilidad de una disyunción viene dada por

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

Esta regla es más fácil de recordar si consideramos que los casos donde a es válido, junto con los casos donde b es válido, inequívocamente cubren todos los casos donde a y b son válidos; pero la suma de los dos conjuntos de casos cuenta su intersección dos veces, así que necesitamos restar $P(a \wedge b)$.

AXIOMAS DE KOLMOGOROV

A menudo estos tres axiomas se conocen por **axiomas de Kolmogorov** en honor al matemático ruso Andrei Kolmogorov, quien demostró cómo desarrollar el resto de la teoría probabilista de este sencillo fundamento. Nótese que los axiomas sólo tratan con probabilidades *a priori* en vez de probabilidades condicionales; esto es porque ya hemos definido las últimas en términos de la Ecuación (13.1) anterior.

Utilización de los axiomas de probabilidad

Podemos obtener diversos resultados útiles a partir de los axiomas básicos. Por ejemplo, la regla familiar para la negación se obtiene sustituyendo $\neg a$ por b en de la axioma 3, dándonos:

$P(a \vee \neg a) = P(a) + P(\neg a) - P(a \wedge \neg a)$	(por el axioma 3 con $b = \neg a$)
$P(cierto) = P(a) + P(\neg a) - P(falso)$	(por equivalencia lógica)
$1 = P(a) + P(\neg a)$	(por el axioma 2)
$P(\neg a) = 1 - P(a)$	(por álgebra)

La tercera línea de esta deducción es en sí misma un resultado útil y puede extenderse del caso booleano al caso discreto general. Considere D la variable discreta con dominio $\langle d_1 \ldots, d_n \rangle$. Entonces es fácil demostrar (Ejercicio 13.2) que

$$\sum_{i=1}^{n} P(D = d_i) = 1$$

Es decir, cualquier distribución de probabilidad sobre una sola variable debe sumar 1[6]. Es también cierto que cualquier distribución de probabilidad *conjunta* sobre cualquier *conjunto* de variables debe sumar 1: esto puede verse simplemente construyendo una sola megavariable cuyo dominio es el producto cartesiano de los dominios de las variables originales.

Recuerde que cualquier proposición a es equivalente a la disyunción de todos los sucesos atómicos en los que a se verifica; llamaremos a este conjunto de sucesos $\mathbf{e}(a)$. Recuerde también que los sucesos atómicos son mutuamente exclusivos, así que la probabilidad de cualquier conjunción de sucesos atómicos es cero, por el axioma 2. Así, del axioma 3, podemos deducir la siguiente relación sencilla: *La probabilidad de una*

[6] Para variables continuas, la sumatoria se reemplaza por una integral: $\int_{-\infty}^{\infty} P(X = x)\,dx = 1$.

proposición es igual a la suma de las probabilidades de los sucesos atómicos en los que ésta se verifica; esto es,

$$P(a) = \sum_{e_i \in \mathbf{e}(a)} P(e_i) \tag{13.2}$$

Esta ecuación proporciona un método sencillo para calcular la probabilidad de cualquier proposición, dada una distribución conjunta completa que especifique las probabilidades de todos los sucesos atómicos. (*Véase* Sección 13.4.) En secciones subsiguientes obtendremos reglas adicionales para la manipulación de probabilidades. Primero, sin embargo, examinaremos el fundamento de los axiomas.

Por qué los axiomas de la probabilidad son razonables

Los axiomas de la probabilidad pueden verse como la restricción del conjunto de creencias probabilistas que un agente puede considerar. Esto es un poco análogo al caso lógico, donde un agente lógico no puede creer simultáneamente en A, B y $\neg(A \wedge B)$, por ejemplo. Hay, sin embargo, una complicación adicional. En el caso lógico, la definición semántica de conjunción significa que al menos una de las tres creencias mencionadas *debe ser falsa en el mundo*, así es ilógico para un agente creer en las tres. Con probabilidades, por otro lado, las oraciones no se refieren al mundo directamente, sino al estado de conocimiento propio del agente. ¿Por qué, entonces, no puede un agente considerar el siguiente conjunto de creencias, que claramente viola el axioma 3?

$$\begin{array}{ll} P(a) = 0{,}4 & P(a \wedge b) = 0{,}0 \\ P(b) = 0{,}3 & P(a \vee b) = 0{,}8 \end{array} \tag{13.3}$$

Este tipo de cuestión ha sido el tema de décadas de debate intenso entre aquellos que abogan por el uso de las probabilidades como la única forma legítima para los grados de creencia y aquellos que defienden aproximaciones alternativas. Aquí, damos un argumento para los axiomas de la probabilidad, establecidos inicialmente en 1931 por Bruno de Finetti.

La clave para el argumento de de Finetti es la conexión entre grados de creencia y acciones. La idea es que si un agente tiene algún grado de creencia en una proposición a, entonces el agente debería ser capaz de establecer boletos[7] que le son indiferentes a una apuesta a favor o en contra de a. Imagínelo como un juego entre dos agentes: el Agente 1 establece «mi grado de creencia en el suceso a es 0,4», el Agente 2 entonces es libre para elegir apostar tanto a favor como en contra de a, en juegos que son consistentes con el grado de creencia manifestado. Es decir, el Agente 2 podría apostar elegir que a ocurrirá, apostando cuatro dólares contra los seis dólares del Agente 1. O el Agente 2 podría apostar seis dólares contra cuatro dólares de que a no ocurrirá[8]. Si el grado de creencia de un agente no refleja acertadamente el mundo, entonces debería esperar a que

[7] Boletos. El término original inglés es *odds*. En el contexto de apuestas se entiende por *odds* a la proporción en que se ofrece pagar una apuesta y que refleja las posibilidades de acierto de la misma. Por ejemplo, en una expresión como «el apostante está ofreciendo diez contra uno», el boleto sería «diez contra uno». *(Nota del traductor.)*

[8] Uno puede argumentar que las preferencias del agente para saldos diferentes son tales que la posibilidad de la pérdida de un dólar no está contrarestada a una posibilidad igual para la ganancia de un dólar. Una posible respuesta es hacer las cantidades apostadas lo suficientemente pequeñas hasta evitar este problema. El análisis de Savage (1954) salva la cuestión totalmente.

el agente tienda a perder dinero a lo largo de la ejecución ante un agente enemigo cuyas creencias reflejan más fielmente el estado del mundo.

Pero de Finetti demostró algo más fuerte: *Si el Agente 1 expresa un conjunto de grados de creencia que viola los axiomas de la teoría de la probabilidad entonces hay una combinación de apuestas para el Agente 2 que* **garantiza** *que el Agente 1 perderá dinero* **todas** *las veces*. Así, si acepta la idea de que un agente estaría dispuesto a «poner su dinero donde están sus probabilidades.», entonces debería aceptar que es irracional tener creencias que violen los axiomas de la probabilidad.

Uno pudiera imaginar que este juego de apuestas es más artificioso. Por ejemplo, ¿qué pasa si uno rehusara apostar?, ¿eso pone fin a la discusión? La respuesta es que el juego de apuestas es un modelo abstracto para la situación de toma de decisiones en la que cada agente está inevitablemente involucrado en todo momento. Cada acción (incluyendo la inactividad) es un tipo de apuesta, y cada consecuencia puede verse como un pago de la apuesta. El rechazo a apostar es como el rechazo a dejar que el tiempo pase.

No presentaremos la demostración del teorema de de Finetti, pero mostraremos un ejemplo. Suponga que el Agente 1 tiene el conjunto de grados de creencia de la Ecuación (13.3). La Figura 13.2 muestra que si el Agente 2 elige apostar cuatro dólares sobre a, tres dólares por b, y dos dólares por $\neg(a \vee b)$, entonces el Agente 1 siempre pierde dinero, independientemente del resultado final para a y b.

Agente 1		Agente 2		Consecuencia para el Agente 1			
Proposición	Creencia	Apostar a	Apuesta	$a \wedge b$	$a \wedge \neg b$	$\neg a \wedge b$	$\neg a \wedge \neg b$
a	0,4	a	4 a 6	–6	–6	4	4
b	0,3	b	3 a 7	–7	3	–7	3
$a \vee b$	0,8	$\neg(a \vee b)$	2 a 8	2	2	2	–8
				–11	–1	–1	–1

Figura 13.2 Como el Agente 1 tiene creencias inconsistentes, el Agente 2 es capaz de concebir un conjunto de apuestas que garantizan la pérdida para el Agente 1, no importa cuál sea el resultado de a y b.

Otros argumentos filosóficos consistentes se han postulado para el uso de probabilidades, destacando los de Cox (1946) y Carnap (1950). Sin embargo, estando la vida por el camino que está, las demostraciones prácticas a veces hablan más alto que las demostraciones. El éxito de los sistemas de razonamiento basados en la teoría de la probabilidad ha sido mucho más efectivo para la generación de conversos. Ahora veremos cómo los axiomas pueden utilizarse para hacer inferencias.

13.4 Inferencia usando las distribuciones conjuntas totales

INFERENCIA PROBABILISTA

En esta sección describiremos un método simple para **inferencia probabilista** (es decir, el cálculo de probabilidades posteriores para proposiciones-pregunta a partir de la

evidencia observada. Usaremos la distribución conjunta completa como la «base de conocimiento» desde la que se deducirán las respuestas de todas las preguntas. Durante el desarrollo también introduciremos diferentes técnicas útiles para la manipulación de ecuaciones que involucren probabilidades.

Comenzamos con un ejemplo muy sencillo: un dominio consta exactamente de tres variables booleanas *Dolor-de-muelas*, *Caries* e *Infectarse* (la sonda sucia de acero del dentista me infectó los dientes). La distribución conjunta completa es una tabla de dimensión $2 \times 2 \times 2$ como se muestra en la Figura 13.3.

	dolor-de-muelas		$\neg$*dolor-de-muelas*	
	infectarse	$\neg$*infectarse*	*infectarse*	$\neg$*infectarse*
caries	0,108	0,012	0,072	0,008
$\neg$*caries*	0,016	0,064	0,144	0,576

Figura 13.3 Una distribución conjunta completa para el mundo *Dolor-de-muelas*, *Caries* e *Infectarse*

Nótese que las probabilidades de la distribución conjunta suman 1, como se requiere por los axiomas de la probabilidad. Nótese también que la Ecuación (13.2) nos da un modo directo para calcular la probabilidad de cualquier proposición, simple o compleja: simplemente identificamos aquellos sucesos atómicos en los que la proposición es cierta y sumamos sus probabilidades. Por ejemplo, hay seis sucesos atómicos en los que se verifica *caries* $\vee$ *dolor-de-muelas*:

$$\begin{aligned} P(caries \vee dolor\text{-}de\text{-}muelas) &= 0{,}108 + 0{,}012 + 0{,}072 + 0{,}008 + 0{,}016 + 0{,}064 \\ &= 0{,}28 \end{aligned}$$

Una operación particular muy común es obtener la distribución definida sobre algún subconjunto de variables o una sola variable. Por ejemplo, sumando las entradas de la primera fila se obtiene la **probabilidad marginal** [9] o incondicional de *caries*:

PROBABILIDAD MARGINAL

$$P(caries) = 0{,}108 + 0{,}012 + 0{,}072 + 0{,}008 = 0{,}2$$

MARGINALIZACIÓN

Este proceso es conocido como **marginalización**, o **sumatoria de eliminación**, porque las variables diferentes a *Caries* son sumadas para eliminarlas. Podemos escribir la siguiente regla de marginalización general para cualquier conjunto de variables **Y** y **Z**:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{z}) \tag{13.4}$$

Es decir, una distribución sobre **Y** puede obtenerse a partir de cualquier distribución conjunta que la contenga sumando en todas las demás variables. Una variante de esta regla involucra probabilidades condicionales en vez de probabilidades conjuntas usando la regla del producto:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}|\mathbf{z})P(\mathbf{z}) \tag{13.5}$$

[9] Así llamada debido a una práctica común entre los actuarios de escribir las sumas de las frecuencias observadas en el margen de las tablas de seguros.

CONDICIONAMIENTO

Esta regla se conoce como **condicionamiento**. La marginalización y el condicionamiento entrarán en acción como reglas útiles para todo tipo de deducciones que incluyan expresiones probabilistas.

En la mayoría de los casos, estaremos interesados en el cálculo de probabilidades *condicionales* de algunas variables, dada la evidencia sobre otras. Las probabilidades condicionales pueden hallarse usando primero la Ecuación (13.1) para obtener una expresión en términos de probabilidades incondicionales y entonces calcular la expresión a partir de la distribución conjunta completa. Por ejemplo, podemos calcular la probabilidad de una caries, dada la evidencia de dolor de muelas, como sigue:

$$\begin{aligned} P(caries|dolor\text{-}de\text{-}muelas) &= \frac{P(caries \wedge dolor\text{-}de\text{-}muelas)}{P(dolor\text{-}de\text{-}muelas)} \\ &= \frac{0{,}108 + 0{,}012}{0{,}108 + 0{,}012 + 0{,}016 + 0{,}064} = 0{,}6 \end{aligned}$$

Para verificarlo, calculamos también la probabilidad de que no hay caries, dado un dolor de muelas:

$$\begin{aligned} P(\neg caries|dolor\text{-}de\text{-}muelas) &= \frac{P(\neg caries \wedge dolor\text{-}de\text{-}muelas)}{P(dolor\text{-}de\text{-}muelas)} \\ &= \frac{0{,}016 + 0{,}064}{0{,}108 + 0{,}012 + 0{,}016 + 0{,}064} = 0{,}4 \end{aligned}$$

Nótese que en estos dos cálculos el término $1/P(dolor\text{-}de\text{-}muelas)$ permanece constante, sin importar qué valor de *Caries* calculamos. De hecho, puede verse como una constante de **normalización** para la distribución $\mathbf{P}(Caries|dolor\text{-}de\text{-}muelas)$, asegurando que la suma total dará 1. En todos los capítulos que traten con probabilidades, usaremos α para denotar esas constantes. Con esta notación, podemos escribir las dos ecuaciones previas en una:

NORMALIZACIÓN

$$\begin{aligned} \mathbf{P}(Caries/dolor\text{-}de\text{-}muelas) &= \alpha\, \mathbf{P}(Caries, dolor\text{-}de\text{-}muelas) \\ &= \alpha\, [\mathbf{P}(Caries, dolor\text{-}de\text{-}muelas, infectarse) + \\ &\qquad \mathbf{P}(Caries, dolor\text{-}de\text{-}muelas, \neg infectarse)] \\ &= \alpha\, [\langle 0{,}108, 0{,}016\rangle + \langle 0{,}012, 0{,}064\rangle] = \alpha\, \langle 0{,}12, 0{,}08\rangle = \langle 0{,}6, 0{,}4\rangle. \end{aligned}$$

La normalización funciona como un atajo útil en muchos cálculos probabilistas.

Del ejemplo, podemos extraer un procedimiento de inferencia general. Seguiremos por el caso en el que la pregunta abarca a una sola variable. Necesitaremos alguna notación: sea X la variable pregunta (*Caries* en el ejemplo), sea $\mathbf{E}$ el conjunto de variables evidencia (como *Dolor-de-muelas* en el ejemplo), sea $\mathbf{e}$ sus valores observados, y sea $\mathbf{Y}$ las variables no observadas restantes (como *Infectarse* en el ejemplo). La pregunta es $\mathbf{P}(X|\mathbf{e})$ y puede evaluarse como

$$\mathbf{P}(X|\mathbf{e}) = \alpha\, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}), \tag{13.6}$$

donde la sumatoria es sobre todas las posibles $\mathbf{y}$s (es decir, todas la combinaciones de valores posibles de las variables no observadas $\mathbf{Y}$). Nótese que el agrupamiento de las variables X, $\mathbf{E}$, e $\mathbf{Y}$ constituyen el conjunto completo de variables para el dominio, así $\mathbf{P}(X,$

e, **y**) es simplemente un subconjunto de probabilidades de la distribución conjunta completa. El algoritmo se muestra en la Figura 13.4. El ciclo sobre los valores de X y los valores de Y es para contar todos los posibles sucesos atómico con **e** fijo, se suman todas sus probabilidades de la tabla conjunta, y se normalizan los resultados.

```
función PREGUNTAR-CONTAR-CONJUNTA (X, e, P) retorna una distribución sobre X
  entradas: X, la variable pregunta
            e, los valores observados de las variables E
            P, una distribución conjunta sobre las variables {X} ∪ E ∪ Y
                                                    / * Y = variables ocultas * /

  Q(X) ← una distribución sobre X, inicialmente está vacía
  para cada valor x_i de X hacer
    Q(x_i) ← CONTAR-CONJUNTA(x_i, e, Y, [ ], P)
  retorna NORMALIZAR (Q(X))
```

```
función CONTAR-CONJUNTA(x, e, vars, valores, P) retorna un número real
  si VACIO?(vars) entonces retorna P(x, e, valores)
  Y ← PRIMERO(vars)
  retorna Σ_y CONTAR-CONJUNTA(x, e, RESTO(vars), [y|valores], P)
```

Figura 13.4 Un algoritmo para inferencia probabilista basada en contar las entradas de una sistribución conjunta completa.

Dada la distribución conjunta completa con la que operar, PREGUNTAR-CONTAR-CONJUNTA es un algoritmo completo para la respuesta de preguntas probabilistas con variables discretas. Sin embargo, no se puede generalizar bien: para un dominio representado por n variables booleanas, requiere una tabla de entrada de tamaño $O(2^n)$ y necesita un tiempo de orden $O(2^n)$ para procesar la tabla. En un problema realista, estaríamos considerando cientos o miles de variables, y no sólo tres. Enseguida se hace completamente impracticable definir el enorme número de probabilidades que se requieren (la experiencia necesaria para estimar por separado cada una de las entradas de la tabla sencillamente no puede existir).

Por estas razones, la distribución conjunta completa en forma de tabla no es una herramienta práctica para la construcción de sistemas de razonamiento (aunque las notas históricas del final del capítulo incluyen una aplicación del mundo real con este método). Más bien debe verse como el fundamento teórico con el que pueden construirse aproximaciones más efectivas. El resto de este capítulo introduce algunas de las ideas básicas necesarias como los preparativos para el desarrollo de sistemas realistas en el Capítulo 14.

13.5 Independencia

Extendamos la distribución conjunta completa de la Figura 13.3 añadiendo una cuarta variable, *Tiempo*. La distribución conjunta completa entonces se convierte en **P**(*Dolor-*

de-muelas, *Infectarse*, *Caries*, *Tiempo*), que tiene 32 entradas (ya que *Tiempo* tiene cuatro valores). Está formada por cuatro «ediciones» de la tabla que se muestra en la Figura 13.3, una por cada tipo de tiempo. Parece natural preguntar cómo se relaciona cada una de estas ediciones con las otras y con la tabla original de tres variables. Por ejemplo, ¿cómo están relacionados $P(dolor\text{-}de\text{-}muelas, infectarse, caries, Tiempo = nublado)$ y $P(dolor\text{-}de\text{-}muelas, infectarse, caries)$? Un modo de responder a esta pregunta es usar la regla del producto:

$$\begin{aligned} &P(dolor\text{-}de\text{-}muelas, infectarse, caries, Tiempo = nublado) \\ &\quad = P(Tiempo = nublado \mid dolor\text{-}de\text{-}muelas, infectarse, caries) \\ &\qquad\qquad P(dolor\text{-}de\text{-}muelas, infectarse, caries) \end{aligned}$$

Ahora, a menos que sea un dios, no debería imaginar que los problemas dentales de uno influyen en el tiempo. Así, la siguiente oración parece razonable:

$$P(Tiempo = nublado \mid dolor\text{-}de\text{-}muelas, infectarse, caries) = P(Tiempo = nublado) \quad (13.7)$$

De aquí, podemos deducir que

$$\begin{aligned} &P(dolor\text{-}de\text{-}muelas, infectarse, caries, Tiempo = nublado) \\ &\qquad = P(Tiempo = nublado)\, P(dolor\text{-}de\text{-}muelas, infectarse, caries) \end{aligned}$$

Una ecuación similar existe para *cada entrada* de $\mathbf{P}(Dolor\text{-}de\text{-}muelas, Infectarse, Caries, Tiempo)$. De hecho, podemos escribir la ecuación general:

$$\begin{aligned} &\mathbf{P}(Dolor\text{-}de\text{-}muelas, Infectarse, Caries, Tiempo) \\ &\qquad = \mathbf{P}(Dolor\text{-}de\text{-}muelas, Infectarse, Caries)\, \mathbf{P}(Tiempo) \end{aligned}$$

Así, la tabla de 32 elementos para cuatro variables puede construirse a partir de una tabla de ocho elementos y una tabla de cuatro elementos. Esta descomposición se ilustra esquemáticamente en la Figura 13.5(a).

Figura 13.5 Dos ejemplos de factorización de una distribución conjunta grande en distribuciones más pequeñas usando la independencia absoluta. (a) Los problemas del tiempo y dentales son independientes. (b) Los lanzamientos de las monedas son independientes.

INDEPENDENCIA

La propiedad que usamos en la Ecuación (13.7) se llama **independencia** (también llamada **independencia marginal** e **independencia absoluta**). En particular, el tiempo es independiente de los problemas dentales de uno. La independencia entre las proposiciones a y b pueden escribirse como:

$$P(a|b) = P(a) \quad \text{o} \quad P(b|a) = P(b) \quad \text{o} \quad P(a \wedge b) = P(a)P(b) \tag{13.8}$$

Todas estas fórmulas son equivalentes (Ejercicio 13.7). La independencia entre las variables X e Y puede escribirse como sigue (de nuevo, todas éstas son equivalentes):

$$\mathbf{P}(X|Y) = \mathbf{P}(X) \quad \text{o} \quad \mathbf{P}(Y|X) = \mathbf{P}(Y) \quad \text{o} \quad \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y)$$

Las afirmaciones de independencia están basadas usualmente en el conocimiento del dominio. Como hemos visto, pueden reducir drásticamente la cantidad de información necesaria para especificar la distribución conjunta completa. Si el conjunto completo de variables puede dividirse en subconjuntos independientes, entonces la conjunta completa puede ser *factorizada* en distribuciones conjuntas separadas sobre esos subconjuntos. Por ejemplo, la distribución conjunta del resultado de n lanzamientos de moneda independientes, $\mathbf{P}(C_1, \ldots, C_n)$, puede representarse como el producto de n distribuciones de una sola variable $\mathbf{P}(C_i)$. En una línea más práctica, la independencia de la odontología y la meteorología es un buen hecho, ya que de otro modo la práctica de la odontología requeriría un conocimiento profundo de la meteorología y *viceversa*.

Así, cuando están disponibles, entonces, las afirmaciones de independencia, nos pueden ayudar para la reducción del tamaño de la representación del dominio y la complejidad del problema de inferencia. Desafortunadamente, la separación nítida de conjuntos enteros de variables por independencia es bastante rara. Siempre que exista una conexión entre dos variables, aunque sea indirecta, la independencia no se verificará. Además, incluso subconjuntos independientes pueden ser bastantes grandes, por ejemplo, la odontología cubre docenas de enfermedades y cientos de síntomas, todos ellos interrelacionados. Para manejar tales problemas, necesitaremos más métodos ingeniosos que el concepto simple de independencia.

13.6 La regla de Bayes y su uso

En el Aparatado 13.2, definimos la **regla del producto** y señalamos que puede escribirse de dos formas por la conmutatividad de la conjunción:

$$P(a \wedge b) = P(a|b)P(b)$$
$$P(a \wedge b) = P(b|a)P(a)$$

Igualando los miembros derechos y dividiendo por $P(a)$, obtenemos

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} \tag{13.9}$$

REGLA DE BAYES

Esta ecuación se conoce como **regla de Bayes** (también como ley de Bayes o teorema de Bayes). Esta ecuación sencilla subyace en todos los sistemas de IA modernos con in-

ferencia probabilística. El caso más general para variables multivaluadas puede escribirse con la notación **P** como

$$\mathbf{P}(Y|X) = \frac{\mathbf{P}(X|Y)\mathbf{P}(Y)}{\mathbf{P}(X)}$$

donde de nuevo se interpreta como la representación de un conjunto de ecuaciones, y cada relación con valores específicos de las variables. También tendremos ocasión de usar una versión más general, condicionada a alguna evidencia base **e**.

$$\mathbf{P}(Y|X, \mathbf{e}) = \frac{\mathbf{P}(X|Y, \mathbf{e})\mathbf{P}(Y|\mathbf{e})}{\mathbf{P}(X|\mathbf{e})} \tag{13.10}$$

Aplicación de la regla de Bayes: el caso sencillo

A primera vista, la regla de Bayes no parece muy útil. Requiere tres términos (una probabilidad condicional y dos probabilidades incondicionales) sólo para calcular una probabilidad.

La regla de Bayes es útil en la práctica porque hay muchos casos donde disponemos de buenas estimaciones probabilistas para estos tres números y necesitamos calcular el cuarto. En una tarea como el diagnóstico médico, con frecuencia tenemos probabilidades condicionales de relaciones causales y necesitamos deducir una diagnóstico. Un médico sabe que la enfermedad de meningitis causa al paciente que el cuello se agarrote, digamos, el 50 por ciento de las veces. El médico también conoce algunos hechos incondicionados: la probabilidad *a priori* de que un paciente tenga meningitis es 1/50.000 y la probabilidad priori de que cualquier paciente tenga un cuello agarrotado es 1/20. Si tomamos s como la proposición de que el paciente tiene un cuello agarrotado y m como la proposición de que el paciente tiene meningitis, tenemos

$$\begin{aligned} P(s|m) &= 0{,}5 \\ P(m) &= 1/50.000 \\ P(s) &= 1/20 \\ P(m|s) &= P(s|m)P(m)\backslash P(s) = \frac{0{,}5 \times 1/50.000}{1/20} = 0{,}0002 \end{aligned}$$

Es decir, esperamos que sólo un paciente de entre 5.000 con un cuello agarrotado tenga meningitis. Nótese que, aun pensando en que una meningitis apunta a un cuello agarrotado con bastante contundencia (con probabilidad 0,5), la probabilidad de meningitis en el paciente se queda pequeña. Esto es porque la probabilidad a priori de un cuello agarrotado es mucho más alta que la de meningitis.

La Sección 13.4 ilustró un proceso por el que se puede evitar la asignación de probabilidades de la evidencia (aquí, $P(s)$) consistente en el cálculo de la probabilidad *a posteriori* para cada variable de la variable pregunta (aquí, m y $\neg m$) y entonces normalizar el resultado. El mismo proceso puede aplicarse en la regla de Bayes. Tenemos

$$\mathbf{P}(M|s) = \alpha \langle P(s|m)P(m), P(s|\neg m)P(\neg m)\rangle$$

Así, para usar esta aproximación necesitamos estimar $P(s|\neg m)$ en vez de $P(s)$. A veces es más fácil, a veces es más difícil. La fórmula general de la regla de Bayes con normalización es

$$\mathbf{P}(Y|X) = \alpha\, \mathbf{P}(X|Y)\mathbf{P}(Y), \tag{13.11}$$

donde α es la constante de normalización necesaria para hacer que las entradas de $\mathbf{P}(Y|X)$ sumen 1.

Una cuestión obvia sobre la regla de Bayes es por qué uno debe disponer de la probabilidad condicional en un sentido, pero no en el otro. En el dominio de la meningitis, quizás el médico sabe que un cuello agarrotado implica meningitis en uno de cada 5.000 casos; es decir, el médico tiene información cuantitativa en el sentido de los síntomas a las causas, el sentido **diagnóstico**. Tal médico no tiene necesidad de usar la regla de Bayes. Desafortunadamente, *el conocimiento diagnóstico es con frecuencia más frágil que el conocimiento causal*. Si hay una epidemia repentina de meningitis, la probabilidad incondicional de meningitis, $P(m)$, aumentará. El médico que obtuvo la probabilidad diagnóstica $P(m|s)$ directamente de la observación estadística de pacientes antes de la epidemia no tendrá ni idea de cómo actualizar el valor, pero el médico que calcula $P(m|s)$ a partir de los otros tres valores verá que $P(m|s)$ aumentará proporcionalmente a $P(m)$. Lo que es más importante, la información causal $P(s|m)$ no se ve *afectada* por la epidemia, ya que sencillamente refleja el modo en que actúa la meningitis. El uso de este tipo de conocimiento causal directo o basado en modelos proporciona la robustez decisiva necesaria que hace viables a los sistemas probabilistas en el mundo real.

Utilización de la regla de Bayes: combinación de evidencia

Hemos visto que la regla de Bayes puede ser un elemento de evidencia útil para la contestación de preguntas probabilistas condicionales (por ejemplo, el cuello agarrotado). En particular, hemos argumentado que la información probabilista está muchas veces disponible de la forma $P(efecto|causa)$. ¿Qué ocurre cuando tenemos dos o más elementos de evidencia? Por ejemplo, ¿qué puede concluir una dentista si su sonda sucia de acero infecta los dientes dolorosos de un paciente? Si conocemos la distribución conjunta completa (Figura 13.3), uno puede leer la respuesta:

$$\mathbf{P}(Caries|dolor\text{-}de\text{-}muelas \wedge infectarse) = \alpha \langle 0{,}108, 0{,}016\rangle \approx \langle 0{,}871, 0{,}129\rangle$$

Sabemos, sin embargo, que tal metodología no es ampliable a un número más amplio de variables.

Podemos intentar la utilización de la regla de Bayes para reformular el problema:

$$\begin{aligned}&\mathbf{P}(Caries|dolor\text{-}de\text{-}muelas \wedge infectarse)\\ &\qquad = \alpha\mathbf{P}(dolor\text{-}de\text{-}muelas \wedge infectarse|Caries)\mathbf{P}(Caries)\end{aligned} \tag{13.12}$$

Para que esta reformulación funcione, necesitamos conocer las probabilidades condicionales de la conjunción *dolor-de-muelas* $\wedge$ *infectarse* para cada valor de *Caries*. Esto podría ser factible para sólo dos variables de evidencia, pero de nuevo no será ampliable. Si hay n posibles variables de evidencia (rayos X, dieta, higiene bucal, etc.), entonces hay 2^n posibles combinaciones de valores observados para los que necesitaríamos conocer las probabilidades condicionales. Podríamos también volver a utilizar la distribución conjunta completa. Esto es lo que primero condujo a los investigadores apartados de la teoría de la probabilidad hacia métodos aproximados para la combinación de evi-

dencia que, mientras daban respuestas erróneas, requieren pocos números para dar cualquier respuesta a todo.

Mejor que tomar este camino, necesitamos encontrar algunas afirmaciones adicionales sobre el dominio que nos permitirá simplificar las expresiones. La noción de **independencia** de la Sección 13.5 proporciona una pista, pero necesita refinarse. Sería precioso si *Dolor-de-muelas* e *Infectarse* fueran independientes, pero no lo son: si la sonda infecta la muela, probablemente tiene una caries y eso probablemente causa un dolor de muelas. Sin embargo, estas variables *son independientes dada la presencia o la ausencia de caries*. Cada una está directamente causada por la caries, pero ninguna de ellas tiene un efecto directo sobre la otra: el dolor de muelas depende del estado de los nervios de la muela, mientras que la precisión de la sonda depende de la habilidad del dentista, para la cual es irrelevante el dolor de muelas[10]. Matemáticamente, esta propiedad se escribe como

$$\mathbf{P}(\textit{dolor-de-muelas} \wedge \textit{infectarse} | \textit{Caries}) = \mathbf{P}(\textit{dolor-de-muelas} | \textit{Caries})\, \mathbf{P}(\textit{infectarse} | \textit{Caries}) \tag{13.13}$$

INDEPENDENCIA CONDICIONAL

Esta ecuación expresa la **independencia condicional** de *dolor-de-muelas* e *infectarse* dada la *Caries*. Podemos sustituirla en la Ecuación (13.12) para obtener la probabilidad de una caries:

$$\mathbf{P}(\textit{Caries} | \textit{dolor-de-muelas} \wedge \textit{infectarse}) = \alpha \mathbf{P}(\textit{dolor-de-muelas} | \textit{Caries})\, \mathbf{P}(\textit{infectarse} | \textit{Caries}) \mathbf{P}(\textit{Caries}).$$

Ahora los requerimientos de información son los mismos que para la inferencia que utiliza cada elemento de evidencia por separado: la probabilidad *a priori* $\mathbf{P}(\textit{Caries})$ para la variable-pregunta y la probabilidad condicional de cada efecto, dada su causa.

La definición general de independencia condicional de dos variables X e Y, dada una tercera variable Z es

$$\mathbf{P}(X, Y | Z) = \mathbf{P}(X | Z) \mathbf{P}(Y | Z)$$

En el dominio odontólogo, por ejemplo, parece razonable imponer independencia condicional a las variables *Dolor-de-muelas* e *Infectarse*, dada *Caries*:

$$\mathbf{P}(\textit{Dolor-de-muelas}, \textit{Infectarse} | \textit{Caries}) = \mathbf{P}(\textit{Dolor-de-muelas} | \textit{Caries})\, \mathbf{P}(\textit{Infectarse} | \textit{Caries}) \tag{13.14}$$

Nótese que esta afirmación es un poco más extraña que la Ecuación (13.13), que afirma independencia sólo para valores específicos de *Dolor-de-muelas* e *Infectarse*. Como con la independencia absoluta de la Ecuación (13.8), las fórmulas equivalentes

$$\mathbf{P}(X | Y, Z) = \mathbf{P}(X | Z) \quad \text{y} \quad \mathbf{P}(Y | X, Z) = \mathbf{P}(Y | Z)$$

pueden también utilizarse.

La Sección 13.5 mostró que las afirmaciones de independencia absoluta permiten una descomposición de la distribución conjunta completa en elementos mucho más pequeños. Resulta que lo mismo es cierto para afirmaciones de independencia condicio-

[10] Asumimos que el paciente y el dentista son individuos distintos.

nal. Por ejemplo, dada la afirmación de la Ecuación (13.14), podemos deducir una descomposición como sigue:

$$\begin{aligned}&\mathbf{P}(\textit{Dolor-de-muelas, Infectarse, Caries})\\ &\quad = \mathbf{P}(\textit{Dolor-de-muelas, Infectarse}|\textit{ Caries})\mathbf{P}(\textit{Caries}) \quad \text{(regla del producto)}\\ &\quad = \mathbf{P}(\textit{Dolor-de-muelas}|\textit{Caries})\mathbf{P}(\textit{Infectarse, Caries})\mathbf{P}(\textit{Caries}) \quad \text{[usando (13.14)]}.\end{aligned}$$

De este modo, la tabla grande original se descompone en tres tablas más pequeñas. La tabla original tiene siete números independientes ($2^3 - 1$, ya que los números deben sumar 1). Las tablas más pequeñas contienen cinco números independientes ($2 \times (2^1 - 1)$ para cada distribución de probabilidad condicional y $2^1 - 1$ para la *a priori* sobre *Caries*). Esto podría parecer que no es un éxito importante, pero el propósito es que, para n síntomas que son todos independientes condicionalmente dada *Caries*, el tamaño de la representación crece en orden $O(n)$ en vez de $O(Z^n)$. Así, *las declaraciones de independencia condicional permiten aumentar los sistemas probabilistas; más aún, aquellas están normalmente mucho más disponibles que las declaraciones de independencia absoluta*. Conceptualmente, *Caries* **separa** *Dolor-de-muelas* e *Infectarse* porque es una causa directa de las dos. La descomposición de grandes dominios probabilistas en subconjuntos conectados débilmente vía independencia condicional es uno de los más importantes desarrollos en la reciente historia de la IA.

SEPARACIÓN

El ejemplo odontólogo ilustra un patrón de ocurrencia común en el que una sola causa influye directamente en un número de efectos, todos ellos independientes condicionalmente, dada la causa. La distribución conjunta completa puede escribirse como:

$$\mathbf{P}(\textit{Causa, Efecto}_1\ldots, \textit{Efecto}_n) = \mathbf{P}(\textit{Causa})\prod_i \mathbf{P}(\textit{Efecto}_i|\textit{Causa})$$

SIMPLE-BAYES

Tal distribución de probabilidad se llama modelo **simple-Bayes** «simple» porque es usado a menudo (como un supuesto simplificado) en casos donde las variables «efecto» *no* son independientes condicionalmente dada la variable causa. (El modelo simple-Bayes es a veces denominado un **clasificador Bayesiano**, uso un tanto descuidado que ha incitado a los bayesianos verdaderos a llamarlo el modelo **idiota-Bayes**.) En la práctica, los sistemas simples-Bayes pueden trabajar sorprendentemente bien, incluso cuando los supuestos de independencia no son ciertos. El Capítulo 20 describe métodos para el aprendizaje de las distribuciones de un simple-Bayes a partir de las observaciones.

IDIOTA-BAYES

13.7 El mundo *wumpus* revisado

Podemos combinar muchas de las ideas de este capítulo para resolver problemas de razonamiento probabilista en el mundo *wumpus*. (*Véase* Capítulo 7 para una descripción completa del mundo *wumpus*.) La incertidumbre aparece en el mundo *wumpus* porque los sensores del agente dan sólo información parcial y local del mundo. Por ejemplo, la Figura 13.6 muestra una situación en la que cada uno de los cuadrados alcanzables ([1,3], [2,2] y [3,1]) podrían contener un pozo. La inferencia lógica pura no puede concluir nada sobre qué cuadrado es más probablemente seguro, así un agente lógico podría estar for-

(a) (b)

Figura 13.6 (a) Después de encontrar brisa tanto en [1,2] como en [2,1], el agente está atascado, no hay un lugar seguro que explorar. (b) División de los cuadrados en *Conocido*, *Borde*, y *Otros*, para una pregunta sobre [1,3].

zado a elegir aleatoriamente. Veremos que un agente probabilista puede hacerlo mucho mejor que un agente lógico.

Nuestra meta será calcular la probabilidad de que cada uno de los tres cuadrados tenga un pozo. (Para los propósitos de este ejemplo, ignoraremos al *wumpus* y al oro.) Las propiedades relevantes del mundo *wumpus* son que (1) un pozo causa brisas en todos los cuadrados vecinos, y (2) cada cuadrado distinto del [1,1] contiene un pozo con probabilidad 0,2. El primer paso es identificar el conjunto de variables aleatorias que necesitamos:

- Como en el caso lógico proposicional, queremos una variables booleana P_{ij} para cada cuadrado que será cierto si y sólo si el cuadrado $[i,j]$ contiene un pozo.
- También tenemos las variables B_{ij} que son ciertas si y sólo si el cuadrado $[i, j]$ está ventoso; incluimos estas variables sólo para los cuadrados observados, en este caso, [1,1], [1,2], y [2,1].

El siguiente paso es especificar la distribución conjunta completa, $\mathbf{P}(P_{1,1}\ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$. Aplicando la regla del producto, tenemos:

$$\mathbf{P}(P_{1,1}\ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) = \mathbf{P}(B_{1,1}, B_{1,2}, B_{2,1} \mid P_{1,1}\ldots, P_{4,4})\mathbf{P}(P_{1,1}, \ldots, P_{4,4})$$

Esta descomposición hace muy fácil ver cuáles deberían ser los valores de la probabilidad conjunta. El primer término es la probabilidad condicional de una configuración de brisas, dada una configuración de pozos; éste es 1 si las brisas son adyacentes a los pozos y 0 en otro caso. El segundo término es la probabilidad *a priori* de una configuración de pozos. Cada cuadrado contiene un pozo con probabilidad 0,2, independientemente de los otros cuadrados; así,

$$\mathbf{P}(P_{1,1}\ldots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathbf{P}(P_{i,j}) \tag{13.15}$$

Para una configuración con n pozos, ésta es exactamente $0{,}2^n \times 0{,}8^{16-n}$.

En la situación de la Figura 13.6(a), la evidencia consiste en la brisa observada (o su ausencia) en cada cuadrado que es visitado, combinada con el hecho de que cada uno de tales cuadrados no contiene pozo. Abreviaremos estos hechos como $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$ y $conocido = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$. Estamos interesados en responder a preguntas del tipo $\mathbf{P}(P_{1,3}|conocido, b)$: ¿cómo de probable es que [1,3] contenga un pozo, dadas las observaciones que se tienen hasta ahora?

Para responder a esta pregunta, podemos seguir la aproximación estándar propuesta por la Ecuación (13.6) e implementada en PREGUNTAR-CONTAR-CONJUNTA, a saber, sumar las entradas de la distribución conjunta completa. Sea *Desconocido* una variable compuesta formada por las variables $P_{i,j}$ de los cuadrados que no son ni de los cuadrados de *Conocido* ni del cuadrado pregunta [1,3]. Entonces, por la Ecuación (13.6), tenemos

$$\mathbf{P}(P_{1,3}|conocido, b) = \alpha \sum_{desconocido} \mathbf{P}(P_{1,3}, desconocido, conocido, b)$$

Las probabilidades conjuntas completas ya han sido especificadas, es decir, a no ser que nos preocupemos por el cómputo. Hay 12 cuadrados desconocidos; así, la sumatoria contiene $2^{12} = 4.096$ términos. En general, la sumatoria crece exponencialmente con el número de cuadrados.

La intuición sugiere que aquí estamos fallando en algo. Sin duda uno podría preguntar, ¿no son los otros cuadrados irrelevantes? ¡Los contenidos de [4,4] no afectan si [1,3] tiene un pozo! Efectivamente, esta intuición es correcta. Sea *Borde* las variables (distintas a la variable pregunta) que son adyacentes a los cuadrados visitados, en este caso son [2,2] y [3,1]. Además, sea *Otros* las variables de los demás cuadrados desconocidos; en este caso, hay otros 10 cuadrados desconocidos, como se muestran en la Figura 13.6(b). La comprensión crucial es que las brisas observadas son *independientes condicionalmente* de las otras variables, dado lo conocido, el borde, y las variables pregunta. El resto es, como suele decirse, una cosa pequeña de álgebra.

Para usar la sutileza, manipulamos la fórmula pregunta en un formato en el que las brisas están condicionadas a todas las otras variables, y entonces simplificamos usando independencia condicional:

$$\begin{aligned}
&\mathbf{P}(P_{1,3}|conocido, b) \\
&\quad = \alpha \sum_{desconocido} \mathbf{P}(b|P_{1,3}, conocido, desconocido)\mathbf{P}(P_{1,3}, conocido, desconocido) \\
&\qquad\qquad \text{(por la regla del producto)} \\
&\quad = \alpha \sum_{borde} \sum_{otros} \mathbf{P}(b|conocido, P_{1,3}, borde, otros)\mathbf{P}(P_{1,3}, conocido, borde, otros) \\
&\quad = \alpha \sum_{borde} \sum_{otros} \mathbf{P}(b|conocido, P_{1,3}, borde)\mathbf{P}(P_{1,3}, conocido, borde, otros),
\end{aligned}$$

donde el paso final usa independencia condicional. Ahora, el primer término en esta expresión no depende de las otras variables, así que podemos mover la sumatoria al interior:

$$\begin{aligned}
&\mathbf{P}(P_{1,3}/conocido, b) \\
&\qquad = \alpha \sum_{borde} (b|conocido, P_{1,3}, borde) \sum_{otros} \mathbf{P}(P_{1,3}, conocido, borde, otros)
\end{aligned}$$

Por independencia, como en la Ecuación (13.15), el término *a priori* puede factorizarse, y entonces los términos se pueden reordenar:

$$
\begin{aligned}
&\mathbf{P}(P_{1,3}|conocido, b)\\
&\quad = \alpha \sum_{borde} \mathbf{P}(b|conocido, P_{1,3}, borde) \sum_{otros} \mathbf{P}(P_{1,3})\mathbf{P}(conocido)\mathbf{P}(borde)\mathbf{P}(otros)\\
&\quad = \alpha\, \mathbf{P}(conocido)\mathbf{P}(P_{1,3}) \sum_{borde} \mathbf{P}(b|conocido, P_{1,3}, borde)\mathbf{P}(borde) \sum_{otros} \mathbf{P}(otros)\\
&\quad = \alpha'\, \mathbf{P}(P_{1,3}) \sum_{borde} \mathbf{P}(b|conocido, P_{1,3}, borde)\mathbf{P}(borde),
\end{aligned}
$$

donde el último paso incorpora $P(conocido)$ en la constante de normalización y usa el hecho de que $\sum_{otros} P(otros)$ es igual a 1.

Ahora, hay exactamente cuatro términos en la sumatoria sobre las variables del borde $P_{2,2}$ y $P_{3,1}$. El uso de la independencia y de la independencia condicional ha eliminado completamente a los otros cuadrados de la estimación. Nótese que la expresión $\mathbf{P}(b|conocido, P_{1,3}, borde)$ es 1 cuando el borde es consistente con las observaciones de la brisa y 0 en otro caso. Así, para cada valor de $P_{1,3}$ sumamos *los modelos lógicos* para las variables del borde que son consistentes con los hechos conocidos. (Comparar con la enumeración de los modelos en la Figura 7.5.) Los modelos y sus probabilidades *a priori* asociadas —$P(borde)$— se muestran en la Figura 13.7. Tenemos

$$\mathbf{P}(P_{1,3}|conocido, b) = \alpha'\langle 0{,}2(0{,}04 + 0{,}16 + 0{,}16), 0{,}8(0{,}04 + 0{,}16)\rangle \approx \langle 0{,}31, 0{,}69\rangle$$

Esto es, [1,3] (y [3,1] por simetría) contiene un pozo con aproximadamente una probabilidad del 31 por ciento. Un cálculo similar, que al lector le debería apetecer desarrollar, muestra que [2,2] contiene un pozo con una probabilidad aproximada del 86 por ciento. ¡El agente *wumpus* debería definitivamente evitar [2,2]!

Lo que esta sección ha presentado es que hasta problemas aparentemente complicados pueden formularse con precisión en la teoría de la probabilidad y resolverse con algoritmos sencillos. Para conseguir soluciones *eficientes*, pueden utilizarse las relaciones de independencia y de independencia condicional para simplificar las sumatorias que se necesitan. Estas relaciones a menudo corresponden a nuestro entendimiento natural de cómo el problema debería estar descompuesto. En el capítulo siguiente, des-

Figura 13.7 Modelos consistentes para las variables del borde $P_{2,2}$ y $P_{3,1}$, presentando $P(borde)$ para cada modelo: (a) tres modelos con $P_{1,3} = cierto$ mostrando dos o tres pozos, y (b) dos modelos con $P_{1,3} = falso$ mostrando uno o dos pozos.

arrollaremos representaciones formales para tales relaciones así como algoritmos que funcionan con esas representaciones para realizar eficientemente la inferencia probabilista.

13.8 Resumen

Este capítulo ha argumentado que la probabilidad es el modo correcto para razonar con incertidumbre.

- La incertidumbre aparece ya sea por pereza como por ignorancia. Es ineludible en mundos complejos, dinámicos, o inabordables.
- La incertidumbre significa que muchas de las simplificaciones que son posibles con inferencia deductiva no son válidas.
- Las probabilidades expresan la incompetencia del agente para alcanzar una decisión definitiva respecto a la verdad de una oración. Las probabilidades resumen las creencias del agente.
- Las oraciones probabilistas básicas incluyen **probabilidades *a priori*** y **probabilidades condicionales** sobre proposiciones simples y complejas.
- La **distribución de probabilidad conjunta completa** especifica la probabilidad de cada asignación de valores a todas las variables aleatorias. Usualmente es demasiado grande para crearla o usarla en su forma explícita.
- Los axiomas de la probabilidad restringen las posibles asignaciones de probabilidad a las proposiciones. Un agente que viole los axiomas se comportará irracionalmente en alguna situación.
- Cuando la distribución conjunta completa está disponible, puede usarse para responder a preguntas simplemente sumando las entradas de los sucesos atómicos que correspondan a las proposiciones de la pregunta.
- La **independencia absoluta** entre subconjuntos de variables aleatorias puede permitir que la distribución conjunta completa sea factorizada en distribuciones conjuntas más pequeñas. Esto podría reducir enormemente la complejidad, pero rara vez ocurre en la práctica.
- La **regla de Bayes** permite calcular probabilidades desconocidas a partir de probabilidades condicionales, usualmente en el sentido causal. La aplicación de la regla de Bayes con muchos elementos de evidencia tropezará en general con los mismos problemas de generalización que los de la distribución conjunta completa.
- La **independencia condicional** ocasionada por relaciones causales directas en el dominio puede permitir factorizar la distribución conjunta completa en distribuciones condicionales más pequeñas. El modelo **simple-Bayes** supone la independencia condicional de todos las variables de los efectos, dada una sola variable causa, y crece linealmente con el número de efectos.
- Un agente del mundo-*wumpus* puede calcular probabilidades de aspectos no observados del mundo y los usa para tomar mejores decisiones que las que hace un agente puramente lógico.

Notas bibliográficas e históricas

Aunque los juegos de azar datan por lo menos en torno al 300 a.c., el análisis matemático de boletos y probabilidades es mucho más reciente. Algunos trabajos realizados por Mahaviracarya en la India están fechados aproximadamente por el siglo IX d.c. En Europa, los primeros intentos datan sólo del Renacimiento Italiano, alrededor de 1500 d.c. Los primeros análisis sistemáticos significativos los hizo Girolamo Cardano alrededor de 1565, pero se quedaron sin publicar hasta 1663. Por ese tiempo, el descubrimiento por Blaise Pascal (en correo con Pierre Fermat en 1654) de un modo sistemático de calcular probabilidades asentó por primera vez a las probabilidades como una disciplina matemática. El primer libro publicado sobre probabilidades fue *De Ratiociniis in Ludo Aleae* (Huygens, 1657). Pascal también introdujo las probabilidades condicionales, que se trataron en el libro de Huygens. El reverendo Thomas Bayes (1702-1761) introdujo la regla para el razonamiento con probabilidades condicionales que fue llamada como él en su honor. Fue publicada póstumamente (Bayes, 1763). Kolmogorov (1950, primero publicado en Alemania en 1933) presentó por primera vez la teoría de la probabilidad en un marco de trabajo rigurosamente axiomático. Rényi (1970) más tarde dio una presentación axiomática que tomaba la probabilidad condicional como elemento primario, en vez de la probabilidad absoluta.

Pascal usó la probabilidad en aspectos que requerían tanto la interpretación objetiva, como una propiedad del mundo basada en simetrías o frecuencias relativas, y la interpretación subjetiva, basada en el grado de creencia, el anterior en su análisis de probabilidades en juegos de azar, el posterior en la famosa «apuesta de Pascal» que discute sobre la existencia posible de Dios. Sin embargo, Pascal no comprendió claramente la distinción entre estas dos interpretaciones. La distinción fue primero establecida con claridad por James Bernoulli (1654-1705).

Leibniz introdujo la noción «clásica» de probabilidad como una proporción de recuentos, casos igualmente probables, que fue también utilizada por Bernoulli, aunque adquirió importancia por Laplace (1749-1827). Esta noción es ambigua entre la interpretación frecuencialista y la interpretación subjetiva. Los casos que pueden pensarse como igualmente probables son o por una simetría física natural entre ellos, o simplemente porque no tenemos ningún conocimiento que nos guíe para considerar que uno es más probable que otro. El uso de esta reciente consideración subjetiva para justificar la asignación de probabilidades iguales se conoce como el *principio de indiferencia* (Keynes, 1921).

El debate entre objetivos y subjetivos llegó a ser más intenso en siglo XX. Kolmogorov (1963), R. A. Fisher (1922) y Richard von Mises (1928) eran defensores de la interpretación de la frecuencia relativa. La intepretación de la «tendencia» de Karl Popper (1959, primero publicada en Alemania en 1934) rastrea las frecuencias relativas hasta una simetría física subyacente. Frank Ramsey (1931), Bruno de Finetti (1937), R.T. Cox (1946), Leonard Savage (1954) y Richard Jeffrey (1983) interpretaron las probabilidades como el grado de creencia de individuos específicos. Sus análisis de grado de creencia estaban estrechamente relacionados a las utilidades y al comportamiento, concretamente, a la voluntad para identificar apuestas. Rudolf Carnap, siguiendo a Leibniz y Laplace, propuso un tipo diferente de interpretación subjetiva de probabilidad, no como

cualquier grado de creencia del individuo real, sino como un grado de creencia que un individuo idealizado *debería* tener ante una proposición particular *a*, dada un conjunto de evidencias **e**. Carnap intentó ir más allá que Leibniz y Laplace creando esta noción matemáticamente precisa de grado de **confirmación**, como una relación lógica ente *a* y **e**. El estudio de esta relación estaba destinada a constituir una disciplina matemática llamada **lógica inductiva**, análoga a la lógica deductiva (Carnap, 1948, 1950). Carnap no fue capaz de extender su lógica inductiva mucho más allá del caso proposicional, y Putman (1963) mostró que los problemas fundamentales impedirían una extensión estricta a lenguajes capaces de expresar la aritmética.

CONFIRMACIÓN

LÓGICA INDUCTIVA

El problema de las clases de referencia está estrechamente relacionado al intento de encontrar una lógica inductiva. La propuesta de la elección de la clase de referencia «más precisa» de tamaño suficiente la propuso formalmente Reichenbach (1949). Se han hecho varios intentos, principalmente por Henry Kyburg (1977, 1983), para formular políticas más sofisticadas para evitar varias falacias obvias que surgen con la regla de Reichenbach, pero esas aproximaciones siguen siendo un tanto *ad hoc*. Un trabajo más reciente de Bacchus, Grove, Halpern y Soller (1992) extiende los métodos de Carnap a teorías de primer orden, de este modo esquivan muchas de las dificultades asociadas con el método sencillo de la clase de referencia.

El razonamiento probabilista ha sido usado en IA desde 1960, especialmente en diagnóstico médico. Fue usado no sólo para hacer un diagnóstico a partir de la evidencia disponible, sino también para optar a más problemas y análisis utilizando la teoría del valor de la información (Sección 16.6) cuando la evidencia disponible no era concluyente (Gorry, 1968; Gorry *et al*., 1973). Un sistema funcionó mejor que los expertos humanos en el diagnóstico de dolencias abdominales agudas (de Dombal *et al*., 1974). Sin embargo, estos primeros sistemas bayesianos padecieron una serie de problemas. Ya que carecían del modelo teórico de las condiciones que diagnosticaban, eran vulnerables a datos no representativos que aparecían en situaciones para las que sólo se tenía disponible una muestra pequeña (de Dombat *et al*., 1981). Más elementalmente incluso, ya que carecían de un formalismo conciso (tal como el descrito en el Capítulo 14) para la representación y la utilización de la información de independencia condicional, dependían de la adquisición, almacenamiento y procesamiento de grandes tablas de datos probabilistas. Por estas dificultades, los métodos probabilistas para hacer frente a la incertidumbre cayeron en desgracia en la IA desde el 1970 hasta mediados de 1980. Los desarrollos desde finales de 1980 se describen en el capítulo siguiente.

La representación simple Bayes para distribuciones conjuntas se ha estudiado exhaustivamente en la literatura de reconocimiento de patrones desde 1950 (Duda y Hart, 1973). También se ha usado, a menudo sin darse cuenta, en recuperación de textos, empezando con el trabajo de Maron (1961). Los fundamentos probabilistas de esta técnica, mejor descrita en el Ejercicio 13.18, fueron establecidos por Robertson y Sparck Jones (1976). Domingos y Pazzani (1997) proporcionan una explicación para el éxito sorprendente del razonamiento del simple Bayes incluso en dominios donde las suposiciones de independencia son claramente violadas.

Hay mucho libros buenos introductorios a la teoría de la probabilidad, incluyendo los de Chung (1979) y Ross (1988). Morris DeGroot (1989) ofrece una introducción que combina la probabilidad y la estadística desde el punto de vista bayesiano, así como un

texto más avanzado (1970). El libro de Richard Hamming (1991) da una introducción sofisticada matemática a la teoría de la probabilidad desde el punto de vista de una interpretación de la propensión basada en simetría física. Hacking (1975) y Hald (1990) cubren la historia antigua del concepto de probabilidad. Bernstein (1996) proporciona una entretenida narración general de la historia del riesgo.

EJERCICIOS

13.1 Demuestre a partir de los primeros principios que $P(a|b \wedge a) = 1$.

13.2 Usando los axiomas de la probabilidad, pruebe que cualquier distribución de probabilidad sobre una variable aleatoria discreta debe sumar 1.

13.3 ¿Sería racional para un agente tener las tres creencias $P(A) = 0{,}4$, $P(B) = 0{,}3$, $P(A \vee B) = 0{,}5$? Si es así, ¿qué rango de probabilidades sería racional que el agente tuviera para $A \wedge B$? Invente una tabla como la de la Figura 13.2, y muestre cómo apoya su argumento sobre racionalidad. Entonces haga otra versión de la tabla donde $P(A \vee B) = 0{,}7$. Explique por qué es racional tener esta probabilidad, incluso aunque la tabla muestre un caso de pérdida y tres en los que ni tenga ni pérdida ni beneficios. (*Pista*: ¿a qué está el Agente 1 obligado respecto a la probabilidad de los cuatro casos, especialmente al caso que es una pérdida?)

13.4 Esta cuestión está relacionada con las propiedades de los sucesos atómicos, como se analizó en el Apartado 11.4.

a) Pruebe que la disyunción de todos los posibles sucesos atómicos es lógicamente equivalente a *cierto*. [*Pista:* usar una demostración por inducción sobre el número de variables aleatorias.]

b) Pruebe que cualquier proposición es lógicamente equivalente a la disyunción de los sucesos atómicos que conllevan su verdad.

13.5 Considere el dominio del reparto de cartas de cinco cartas de póker con una baraja estándar de 52 cartas, bajo el supuesto de que el repartidor es imparcial

a) ¿Cuántos sucesos atómicos hay en la distribución de probabilidad conjunta (e.d., cuántas manos de cinco cartas hay)?

b) ¿Cuál es la probabilidad de cada suceso atómico?

c) ¿Cuál es la probabilidad de obtener una escalera real?, ¿y cuatro de un tipo?

13.6 Dada la distribución conjunta completa que se muestra en la Figura 13.3, calcule lo siguiente:

a) $P(dolor\text{-}de\text{-}muelas)$

b) $\mathbf{P}(Caries)$

c) $\mathbf{P}(Dolor\text{-}de\text{-}muelas|caries)$

d) $\mathbf{P}(Caries|dolor\text{-}de\text{-}muelas \vee inferctarse)$.

13.7 Demuestre que las tres formas de independencia de la Ecuación (13.8) son equivalentes.

13.8 Después de su revisión anual, el médico tiene malas y buenas noticias. Las malas noticias es que dio positivo para una enfermedad grave y que el test es exacto al 99 por ciento (la probabilidad de dar positivo cuando tiene la enfermedad es 0,99, como lo es la probabilidad de dar negativo cuando no tiene la enfermedad). Las buenas noticias son que esta es una enfermedad poco común, sorprendentemente en sólo una de 10.000 personas de su edad. ¿Por qué es una buena noticia que la enfermedad sea poco común? ¿Cuál es el riesgo de que tenga en realidad la enfermedad?

13.9 Es bastante útil a menudo considerar el efecto de algunas proposiciones específicas en el contexto de algún marco general en el que la evidencia permanece fija, más que en contextos con ausencia completa de información. Las siguientes cuestiones le piden que demuestre más versiones generales de la regla del producto y la regla de Bayes, con respecto a alguna evidencia de fondo **e**:

a) Pruebe la versión condicionada de la regla del producto general:

$$\mathbf{P}(X, Y|\mathbf{e}) = \mathbf{P}(X|Y,\mathbf{e})\mathbf{P}(Y|\mathbf{e})$$

b) Pruebe la versión condicionada de la regla de Bayes de la Ecuación (13.10).

13.10 Demuestre que la afirmación:

$$\mathbf{P}(A, B|C) = \mathbf{P}(A|C)\mathbf{P}(B|C)$$

es equivalente a cualquiera de estas afirmaciones

$$\mathbf{P}(A|B,C) = \mathbf{P}(A|C) \quad \text{y} \quad \mathbf{P}(B|A,C) = \mathbf{P}(B|C)$$

13.11 Suponga que tiene una bolsa que contiene n monedas no trucadas. Está diciendo que $n - 1$ de esta monedas son normales, con caras por una lado y cruces por el otro, mientras que una moneda es una falsificación, con caras a ambos lados.

a) Suponga que mete la mano en la bolsa, escoge una moneda uniformemente al azar, la lanza, y obtiene una cara. ¿Cuál es la probabilidad (condicional) de que la moneda que eligió sea la moneda falsificada?

b) Suponga que continúa el lanzamiento de la moneda un total de k veces después de escogerla y ve k caras. ¿Ahora cuál es la probabilidad condicional de que eligiera la moneda falsa?

c) Suponga que quiere decidir si la moneda elegida era falsa lanzándola k veces. El procedimiento de decisión retorna FALSIFICADO si todos los k lanzamientos presentan caras, en otro caso retorna NORMAL. ¿Cuál es la probabilidad (incondicional) de que este procedimiento cometa un error?

13.12 En este ejercicio, completará el cálculo de la normalización para el ejemplo de la meningitis. Primero, invente un valor adecuado para $P(S|\neg M)$, y usarlo para calcular los valores no normalizados de $P(M|S)$ y $P(\neg M|S)$ (e.d., ignorando el término $P(S)$ en la expresión de la regla de Bayes). Ahora normalice estos valores para que sumen 1.

13.13 Este ejercicio indaga el modo en el que las relaciones de independencia condicional afectan a la cantidad de información necesaria para cálculos probabilistas:

a) Suponga que deseamos calcular $P(h|e_1,e_2)$ y tenemos información de no independencia condicional. ¿Cuál de los siguientes conjuntos de números son suficientes para el cálculo?

(i) $\mathbf{P}(E_1, E_2)$, $\mathbf{P}(H)$, $\mathbf{P}(E_1|H)$, $\mathbf{P}(E_2|H)$

(ii) $\mathbf{P}(E_1, E_2)$, $\mathbf{P}(H)$, $\mathbf{P}(E_1, E_2|H)$

(iii) $\mathbf{P}(H)$, $\mathbf{P}(E_1|H)$, $\mathbf{P}(E_2|H)$

b) Suponga que sabemos que $\mathbf{P}(E_1|H, E_2) = \mathbf{P}(E_1|H)$ para todos los valores de H, E_1, E_2. ¿Ahora cuál de los tres conjuntos son suficientes?

13.14 Sean X, Y, Z variables aleatorias booleanas. Etiquete las ocho entradas de la distribución conjunta $\mathbf{P}(X, Y, Z)$ desde la a hasta la h. Exprese las afirmaciones de que X e Y son independientes condicionalmente dada Z como un conjunto de ecuaciones relacionando a hasta h. ¿Cuántas ecuaciones *no redundantes* hay?

13.15 (Adaptado de Pearl (1988)). Suponga que es testigo de un accidente nocturno en el que el conductor se da a la fuga que implica a un taxi en Atenas. Todos los taxis de Atenas son azules o verdes. Jura, bajo juramento, que el taxi era azul. Exhaustivas experimentaciones demuestran que, bajo las condiciones de poca iluminación, la distinción entre el azul y el verde es fiable un 75 por ciento. ¿Es posible calcular el color más creíble para el taxi? (*Pista:* Distinguir cuidadosamente entre la proposición de que el taxi *es* azul y la proposición de que *parece* azul.)

¿Y en el caso de que nueve de cada 10 taxis atenienses son verdes?

13.16 (Adaptado de Pearl (1988)). Tres prisioneros, A, B y C, están encerrados en sus celdas. Es de conocimiento popular que uno de ellos será ejecutado al día siguiente y los otros absueltos. Sólo el gobernador sabe cuál de ellos será ejecutado. El prisionero A pide un favor al guardián: «Por favor pregunte al gobernador quién será ejecutado, y entonces dele un mensaje a uno de mis amigos B o C y hazle saber que será absuelto mañana.» El guardián está de acuerdo, y vuelve más tarde y le dice a A que le dio el mensaje de indulto a B.

¿Cuáles son las posibilidades de A para que sea ejecutado, dada esta información? (Responda esto *matemáticamente*, no con agitaciones enérgicas de manos)

13.17 Escriba un algoritmo general para responder a preguntas de la forma $\mathbf{P}(Causa|\mathbf{e})$, utilizando una distribución simple Bayes. Debería asumir que la evidencia $\mathbf{e}$ puede asignar valores a *cualquier subconjunto* de las variables efecto.

13.18 La clasificación de textos es la tarea de asignar a una categoría un documento dado de entre un conjunto fijo de categorías, sobre la base de lo que contiene el texto. Los modelos simples Bayes se usan a menudo para esta tarea. En estos modelos, la variable pregunta es la categoría del documento, y las variables «efecto» son la presencia o ausencia de cada palabra en el lenguaje; se supone que las palabras se encuentran en los documentos independientemente, con frecuencias determinadas por la categoría del documento.

a) Explique con precisión cómo puede construirse uno de tales modelos, dado como «datos de entrenamiento» un conjunto de documentos que han sido asignados a las categorías.

b) Explique con precisión cómo catalogar un nuevo documento.

c) ¿Es razonable el supuesto de independencia? Discútalo.

13.19 En nuestro análisis del mundo *wumpus*, usamos el hecho de que cada cuadrado contiene un pozo con probabilidad 0,2, independientemente de los contenidos de los otros cuadrados. Suponga en vez de eso, exactamente $N/5$ pozos están dispersos uniformemente al azar entre los N cuadrados distintos al [1,1]. ¿Son todavía independientes las variables $P_{i,j}$ y $P_{k,l}$? ¿Cuál es ahora la distribución conjunta $\mathbf{P}(P_{1,1}\ldots, P_{4,4})$? Rehaga los cálculos para las probabilidades de los pozos en [1,3] y [2,2].

14 Razonamiento probabilista

Donde explicamos cómo construir modelos de red para razonar bajo incertidumbre según las leyes de la teoría de la probabilidad.

En el Capítulo 13 se declaró la sintaxis y semántica de la teoría de la probabilidad. Destacamos la importancia de las relaciones de independencia y de independencia condicional en la simplificación de las representaciones probabilistas del mundo. Este capítulo introduce un modo sistemático para representar explícitamente tales relaciones en la forma de **Redes Bayesianas**. Definimos la sintaxis y semántica de estas redes y mostramos cómo pueden utilizarse para capturar conocimiento incierto de un modo natural y eficiente. Después mostramos cómo la inferencia probabilista, aunque computacionalmente intratable en el peor de los casos, puede hacerse eficiente en muchas situaciones prácticas. También describimos diversos algoritmos de inferencia aproximada que son a menudo aplicables cuando la inferencia exacta es inviable. Exploramos procedimientos en los que la teoría de la probabilidad puede aplicarse a mundos con objetos y relaciones, es decir, a representaciones de *primer orden*, en contraposición a la *proposicional*. Finalmente, damos una visión general a aproximaciones alternativas al razonamiento incierto.

14.1 La representación del conocimiento en un dominio incierto

En el Capítulo 13, vimos que la distribución de probabilidad conjunta completa puede responder a cualquier pregunta sobre el dominio, pero puede llegar a ser enormemente intratable cuando el número de variables crece. Además, la especificación de probabi-

lidades para sucesos atómicos es bastante antinatural y puede ser muy difícil a no ser que se tenga disponible una gran cantidad de datos desde la que se deduzcan las estimaciones estadísticas.

También vimos que las relaciones de independencia y de independencia condicional entre variables pueden reducir enormemente el número de probabilidades que se necesitan establecer para definir la distribución de probabilidad conjunta. Esta sección introduce una estructura de datos llamada **red bayesiana**[1] para representar las dependencias entre las variables y para mostrar una descripción escueta de *cualquier* distribución de probabilidad conjunta completa.

RED BAYESIANA

Una red bayesiana es un grafo dirigido en el que cada *nodo* está comentado con información probabilista cuantitativa. La especificación completa es como sigue:

1. Un conjunto de variables aleatorias forman los nodos de la red. Las variables pueden ser discretas o continuas.
2. Un conjunto de enlaces dirigidos o flechas conectan pares de nodos. Si hay una flecha de un nodo X a un nodo Y, se dice que X es un *padre* de Y.
3. Cada nodo X_i tiene una distribución de probabilidad condicionada $P(X_i|Padres(X_i))$ que cuantifica el efecto de los padres del nodo.
4. El grafo no tiene ciclos dirigidos (y así es un grafo acíclico dirigido, o GAD).

La topología de la red (el conjunto de nodos y enlaces) especifica las relaciones de independencia condicional que se tienen en el dominio, de una forma que se precisará dentro de poco. El significado *intuitivo* de una flecha en una red construida correctamente es, habitualmente, que X tiene una *influencia directa* sobre Y. Es generalmente sencillo para un experto del dominio decidir qué influencias directas existen en el área, mucho más sencillo, de hecho, que la especificación de las probabilidades. Una vez que la topología de la red bayesiana está diseñada, necesitamos sólo especificar una distribución de probabilidad condicional para cada variable, dados sus padres. Veremos que la combinación de la topología y las distribuciones condicionales son suficientes para especificar (implícitamente) la distribución conjunta completa para todas las variables.

Recuerde el mundo sencillo descrito en el Capítulo 13, formado por las variables *Dolor-de-muelas*, *Caries*, y *Tiempo*. Argumentamos que *Tiempo* es independiente de las otras variables; además, argumentamos que *Dolor-de-muelas* e *Infectarse* son independientes condicionalmente, dada *Caries*. Estas relaciones se presentan en la estructura de la red bayesiana que se muestra en la Figura 14.1. Formalmente, la independencia condicional de *Dolor-de-muelas* e *Infectarse* dada *Caries* está indicada por la *ausencia* de un enlace entre *Dolor-de-muelas* e *Infectarse*. Intuitivamente, la red representa el hecho de que *Caries* es una causa directa de *Dolor-de-muelas* e *Infectarse*, aunque no exista una relación causal directa entre *Dolor-de-muelas* e *Infectarse*.

Ahora consideremos el siguiente ejemplo, que es un poco más complejo: tiene una alarma antirrobo instalada en casa, es bastante fiable en la detección de un robo, pero

[1] Éste es el nombre más común, pero hay muchos otros, incluyendo **red de creencia**, **red probabilista**, **red causal** y **mapa de conocimiento**. En estadística, el término **modelo gráfico** se refiere a una clase un poco más amplia que incluye a las redes bayesianas. Una extensión de las redes bayesianas llamada **red de decisión** o **diagrama de influencia** se tratará en el Capítulo 16.

Figura 14.1 Una red bayesiana sencilla en la que *Tiempo* es independiente de las otras tres variables y *Dolor-de-muelas* e *Infectarse* son independientes condicionalmente, dada *Caries*.

también responde en ocasiones a pequeños terremotos. (Este ejemplo se debe a Judea Pearl, un residente de Los Angeles, de ahí el gran interés en terremotos.) También tiene dos vecinos, John y Mary, quienes han prometido llamarle al trabajo cuando oigan la alarma. John siempre llama cuando oye la alarma, pero a veces confunde el timbre del teléfono con la alarma con lo que llama también. A Mary, por otro lado, le gusta la música bastante fuerte y a veces no oye para nada la alarma. La red bayesiana para este dominio se presenta en la Figura 14.2.

Por el momento, ignoremos las distribuciones condicionales de la figura y centrémonos en la topología de la red. En el caso de esta red-robo, la topología muestra que robos y terremotos afectan directamente a la probabilidad de que la alarma se dispare, pero tanto la llamada de John como la de Mary dependen sólo de la alarma. Así, la red representa nuestras suposiciones de que ellos no perciben ningún robo directamente, que ellos no perciben terremotos menores, y que ellos no dialogan antes de la llamada.

Nótese que la red no tiene nodos que correspondan a la música alta que está escuchando en ese momento Mary o al timbre del teléfono y la confusión de John. Estos

Figura 14.2 Una red bayesiana típica, mostrando tanto la topología de la red como las tablas de probabilidad condicional (TPCs). En las TPCs, las letras *B, E, A, J* y *M* significan *Robo, Terremoto, Alarma, JohnLlama* y *MaryLlama*, respectivamente.

factores están resumidos en la incertidumbre asociada a los enlaces de *Alarma* hasta *JohnLlama* y *MaryLlama*. Esto muestra tanto la pereza como la ignorancia en funcionamiento: sería mucho trabajo descubrir por qué esos factores serían más o menos posibles en cualquier caso particular, y de todas formas no tenemos un modo razonable para obtener la información relevante. Las probabilidades resumen un conjunto *potencialmente infinito* de circunstancias en las que la alarma podría fallar y no se dispara (humedad alta, fallo de la corriente, baterías agotadas, cables cortados, un ratón muerto en el interior atasca la campana, etc.) o John o Mary podrían fallar y no llaman para informarnos (salir a comer, estar de vacaciones, temporalmente sordos, pasa un helicóptero, etc.). De este modo, un agente pequeño puede hacer frente a un mundo muy grande, al menos aproximadamente. El grado de aproximación puede mejorarse si introducimos información relevante adicional.

TABLA DE PROBABILIDAD CONDICIONAL

CASO DE CONDICIONAMIENTO

Ahora pasemos a las distribuciones condicionales que se muestran en la Figura 14.2. En la figura, cada distribución se muestra como una **tabla de probabilidad condicional**, o TPC. (Este formato de tabla puede usarse para variables discretas; otras representaciones, incluyendo las adecuadas para variables continuas, se describen en la Sección 14.2.) Cada fila de una TPC contiene la probabilidad condicional de cada valor del nodo para un **caso de condicionamiento**. Un caso de condicionamiento es una combinación posible de valores de los nodos padres (un suceso atómico en miniatura, si prefiere). Cada fila debe sumar 1, ya que las entradas representan un conjunto exhaustivo de casos para la variable. Para variables booleanas, una vez que conoce que la probabilidad de un valor verdad es p, la probabilidad de falso debe ser $1 - p$, de esta manera omitimos el segundo número a menudo, como en la Figura 14.2. En general, una tabla para una variable booleana con k padres booleanos contiene 2^k probabilidades explícitas por separado. Un nodo sin padres tiene sólo una fila, que representa las probabilidades *a priori* de cada posible valor de la variable.

14.2 La semántica de las redes bayesianas

La sección anterior describió lo que es una red, pero no lo que significa. Hay dos formas de entender la semántica de las redes bayesianas. La primera es ver la red como una representación de la distribución de probabilidad conjunta. La segunda es verla como una codificación de un conjunto de afirmaciones de independencia condicional. Los dos puntos de vista son equivalentes, pero el primero resulta útil para entender cómo *construir* redes, aunque el segundo es útil para el diseño de procedimientos de inferencia.

La representación de la distribución conjunta completa

Una red bayesiana proporciona una descripción completa del dominio. Cada entrada de la distribución de probabilidad conjunta (de aquí en adelante se abreviará como «conjunta») puede calcularse a partir de la información de la red. Una entrada genérica en la distribución conjunta es la probabilidad de una conjunción de asignaciones concretas a

cada variable, tal como $P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$. Para ésta usaremos la notación abreviada $P(x_1\ldots, x_n)$. El valor de esta entrada está dado por la fórmula

$$\mathbf{P}(x_1\ldots, x_n) = \prod_{i=1}^{n} P(x_i|padres(X_i)), \tag{14.1}$$

donde $padres(X_i)$ denota los valores específicos de las variables de $Padres(X_i)$. Así, cada entrada de la distribución conjunta está representada por el producto de los elementos apropiados de las tablas de las probabilidades condicionales (TPCs) de la red bayesiana. Las TPCs proporcionan así una representación descompuesta de la distribución conjunta.

Para ilustrar esto, podemos calcular la probabilidad de que la alarma ha sonado, pero no ha ocurrido ni un robo ni un terremoto, y tanto John como Mary llaman. Usamos nombres para las variables con una sola letra:

$$\begin{aligned} &P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) \\ &\quad = P(j|a)\, P(m|a)\, P(a|\, \neg b \wedge \neg e)\, P(\neg b)\, P(\neg e) \\ &\quad = 0{,}90 \times 0{,}70 \times 0{,}001 \times 0{,}999 \times 0{,}998 = 0{,}00062. \end{aligned}$$

En la Sección 13.4 se explicó que la distribución conjunta completa puede usarse para responder cualquier pregunta sobre el dominio. Si una red bayesiana es una representación de la distribución conjunta, entonces ésta puede también ser utilizada para responder cualquier pregunta, a partir de la sumatoria de todas las entradas conjuntas relevantes. En la Sección 14.4 se explica cómo hacer esto, aunque también describe métodos que son mucho más eficientes.

Un método para la construcción de redes bayesianas

La Ecuación (14.1) define lo que significa una red bayesiana dada. No explica, sin embargo, cómo *construir* una red bayesiana de tal modo que la distribución conjunta resultante sea una representación buena de un dominio dado. Ahora mostraremos que la Ecuación (14.1) implica ciertas relaciones de independencia condicional que pueden utilizarse para guiar al ingeniero del conocimiento en la construcción de la topología de la red. Primero, reescribimos la distribución conjunta en términos de una probabilidad condicional, usando la regla del producto (*véase* Capítulo 13):

$$P(x_1\ldots, x_n) = P(x_n|x_{n-1}\ldots, x_1)P(x_{n-1}\ldots, x_1)$$

Entonces repetimos el proceso, reduciendo cada probabilidad con conjunciones a una probabilidad condicional con una conjunción más pequeña. Terminamos con un gran producto:

$$\begin{aligned} P(x_1\ldots, x_n) &= P(x_n|x_{n-1}\ldots, x_1)P(x_{n-1}|x_{n-2}\ldots, x_1)\ \ldots P(x_2|x_1)P(x_1) \\ &= \prod_{i=1}^{n} P(x_i|x_{i-1}\ldots, x_1) \end{aligned}$$

REGLA DE LA CADENA

Esta identidad es cierta para cualquier conjunto de variables aleatorias y se llama **regla de la cadena**. Comparándola con la Ecuación (14.1), vemos que la especificación de la distribución conjunta es equivalente a la afirmación general de que, para cada variable X_i de la red,

$$\mathbf{P}(X_i|X_{i-1}\ldots, X_1) = \mathbf{P}(X_i|Padres(X_i)) \tag{14.2}$$

siempre que $Padres(X_i) \subseteq \{X_{i-1}\ldots, X_1\}$. Esta última condición se satisface siempre que se etiqueten los nodos en cualquier orden que sea consistente con el orden parcial implícito en la estructura del grafo.

Lo que la Ecuación (14.2) dice es que la red bayesiana es una representación correcta del dominio sólo si cada nodo es independiente condicionalmente de sus predecesores en la ordenación de nodos, dados sus padres. Así, para construir una red bayesiana con la estructura correcta para el dominio, necesitamos elegir los padres de cada nodo de manera que esta propiedad se verifique. Intuitivamente, los padres de un nodo X_i contendrían a aquellos nodos de $X_1\ldots, X_{i-1}$ que *influyen directamente* en X_i. Por ejemplo, suponga que hemos completado la red de la Figura 14.2 excepto la elección de padres para *MaryLlama*. *MaryLlama* está ciertamente influenciada por si hay un *Robo* o un *Terremoto*, pero no influenciada *directamente*. Intutivamente, nuestro conocimiento del dominio nos dice que estos sucesos influyen en el comportamiento de llamar por teléfono de Mary sólo a través de sus efectos sobre la alarma. Además, dado el estado de la alarma, si John llama no tiene influencia sobre la llamada de Mary. Formalmente hablando, creemos que la afirmación de independencia condicional considera:

$$\mathbf{P}(MaryLlama|JohnLlama, Alarma, Terremoto, Robo) = \mathbf{P}(MaryLlama|Alarma).$$

Compactación y ordenación de nodos

Además de ser una representación completa y no redundante del dominio, una red bayesiana puede a menudo ser mucho más *compacta* que la distribución conjunta completa. Esta propiedad es lo que la hace más factible para manipular dominios con muchas variables. La compactación de redes bayesianas es un ejemplo de una propiedad muy general de los sistemas **estructurados localmente** (también llamados **esparcidos**). En un sistema estructurado localmente, cada subcomponente interactúa directamente con sólo un número limitado de los otros componentes, a pesar del número total de componentes. La estructura local está asociada usualmente con un crecimiento en complejidad lineal más que exponencial. En el caso de redes bayesianas, es razonable suponer que en la mayoría de los dominios cada variable aleatoria está directamente influenciada por al menos otras k, para alguna constante k. Si asumimos n variables booleanas por simplicidad, entonces la cantidad de información necesaria para especificar cada tabla de probabilidad condicional será a lo más 2^k números, y la red completa puede especificarse por $n2^k$ números. En contraposición, la distribución conjunta contiene 2^n números. Para entender este contraste, suponga que tenemos $n = 30$ nodos, cada uno con cinco padres ($k = 5$). Entonces la red bayesiana requiere 960 números, pero la distribución de probabilidad conjunta requiere sobre unos 1.000 millones.

ESTRUCTURADO LOCALMENTE

ESPARCIDO

Hay dominios en los que cada variable puede estar directamente influenciada por todas las demás, de esta manera la red está completamente conectada. Entonces la especificación de las tablas de probabilidad condicional requiere la misma cantidad de información que la especificada en la distribución conjunta. En algunos dominios, habrá dependencias poco firmes que deberían ser estrictamente incluidas, añadiendo un nue-

vo enlace. Pero si estas dependencias son muy indirectas, entonces puede no merecer la pena la complejidad añadida en la red a cambio de un poco de ganancia en la precisión. Por ejemplo, uno podría rebatir a nuestra red-robo sobre la base de que si hay un terremoto, entonces John y Mary no deberían llamar incluso si oyen la alarma, ya que asumen que el terremoto es la causa. Tanto el añadir un enlace desde *Terremoto* hasta *JohnLlama* como a *MaryLlama* (y así agrandamos las tablas) depende de la comparación que se haga entre la importancia de obtener probabilidades más precisas y el costo para especificar la información extra.

Incluso en un dominio estructurado localmente, la construcción de una red bayesiana estructurada localmente no es un problema trivial. Necesitamos no sólo que cada variable esté directamente influenciada por sólo unas pocas de los demás, sino también que la topología de la red refleje aquellas influencias directas con el conjunto apropiado de padres. Debido al modo en que funciona el procedimiento de construcción, los «influyentes directos» tendrán que ser añadidos a la red primero si llegan a ser padres del nodo al que influyen. Así, *el orden correcto en el que agregamos nodos es añadir las «causas raíces» primero, luego las variables que influyen*, y así sucesivamente, hasta llegar a las «hojas», que no tienen influencia causal directa sobre las demás variables.

¿Qué ocurre si por casualidad elegimos el orden equivocado? Consideremos de nuevo el ejemplo del robo. Suponga que decidimos añadir los nodos en el orden *MaryLlama*, *JohnLlama*, *Alarma*, *Robo*, *Terremoto*. Entonces obtenemos la red un poco más complicada que se muestra en la Figura 14.3(a). El proceso es como sigue:

- Añadimos *MaryLlama*: no padres.
- Añadimos *JohnLlama*: si Mary llama, eso probablemente significa que la alarma se ha disparado, lo que por supuesto debería hacer más probable que John llame. Así, *JohnLlama* necesita a *MaryLLama* como padre.
- Añadimos *Alarma*: claramente, si ambos llaman, es más probable que la alarma haya saltado que si uno u otro llama, así necesitamos tanto a *MaryLlama* como *JohnLlama* como padres.
- Añadimos *Robo*: si sabemos el estado de la alarma, entonces la llamada de John o Mary debería darnos información sobre nuestro timbre del telefóno o la música de Mary, pero no sobre el robo:

  $$\mathbf{P}(Robo|Alarma, JohnLlama, MaryLlama) = \mathbf{P}(Robo|Alarma)$$

  Así necesitamos sólo *Alarma* como padre.
- Añadimos *Terremoto*: si la alarma está funcionando, es más creíble que haya habido un terremoto. (La alarma es un detector de terremotos en cierto modo.) Pero si sabemos que ha habido un robo, entonces eso explica la alarma, y la probabilidad de un terremoto debería estar sólo ligeramente por encima de lo normal. Así, necesitamos tanto *Alarma* como *Robo* como padres.

La red resultante tiene dos enlaces más que la red original de la Figura 14.2 y requiere precisar tres probabilidades más. Lo que es peor, algunos de los enlaces representan relaciones indirectas que requieren juicios probabilistas difíciles y antinaturales, tal como la asignación de la probabilidad de *Terremoto*, dado *Robo* y *Alarma*. Este fenómeno es

Figura 14.3 La estructura de la red depende del orden de añadidura. En cada red, hemos introducido nodos en un orden de arriba hacia abajo.

bastante generalizado y está relacionado a la distinción entre los modelos causales y de diagnóstico introducidos en el Capítulo 8. Si intentamos construir un modelo de diagnóstico con enlaces desde los síntomas a las causas (como de *MaryLlama* a *Alarma* o de *Alarma* a *Robo*), acabamos teniendo dependencias adicionales específicas entre causas que hubieran sido independientes (y a menudo, también entre síntomas que se presentan por separado.) *Si nos ceñimos a un modelo causal, terminamos teniendo que especificar menos números, y los números serán muchas veces más fáciles de plantear*. En el dominio de la medicina, por ejemplo, Tversky y Kahneman (1982) han demostrado que los físicos expertos prefieren dar juicios probabilistas para reglas causales antes que para las de diagnóstico.

La Figura 14.3(b) muestra una ordenación de nodos muy mala: *MaryLLama*, *JohnLlama*, *Terremoto*, *Robo*, *Alarma*. Esta red requiere que se especifiquen 31 probabilidades distintas (exactamente las mismas que la distribución conjunta completa). Es importante darse cuenta, sin embargo, de que las tres redes pueden representar *exactamente la misma distribución conjunta*. Las dos últimas versiones simplemente fracasan en la representación de todas las relaciones de independencia condicional y de ahí que terminen especificando muchos números innecesarios.

Relaciones de independencia condicional en redes bayesianas

Hemos proporcionado una semántica «numérica» para redes bayesianas en términos de una representación de la distribución conjunta completa, al igual que la Ecuación (14.1).

Utilizar la semántica para deducir un método de construcción de redes bayesianas nos ha llevado a la consecuencia de que un nodo es independiente condicionalmente de sus predecesores, dados sus padres. Resulta que también podemos ir en la otra dirección. Podemos empezar con una semántica «topológica» que especifique las relaciones de independencia condicional codificadas en la estructura del grafo, y de éstas podemos deducir la semántica «numérica». La semántica topológica viene dada por cualquiera de los requisitos siguientes, que son equivalentes[2]:

DESCENDIENTES

1. Un nodo es independiente condicionalmente de sus no-**descendientes**, dados sus padres. Por ejemplo, en la Figura 14.2, *JohnLlama* es independiente de *Robo* y *Terremoto*, dado el valor de *Alarma*.

MANTO DE MARKOV

2. Un nodo es independiente condicionalmente de todos los demás nodos de la red, dados sus padres, hijos, y padres de sus hijos, esto es, dado su **manto de Markov**. Por ejemplo, *Robo* es independiente de *JohnLlama* y *MaryLlama*, dados *Alarma* y *Terremoto*.

Estos requisitos se ilustran en la Figura 14.4. A partir de estas declaraciones de independencia condicional y las TPCs, la distribución conjunta completa puede reconstruirse; así la semántica «numérica» y la semántica «topológica» son equivalentes.

Figura 14.4 (a) Un nodo X es independiente condicionalmente de sus no-descendientes (por ejemplo, los Z_{ij}) dados sus padres (los U_i que se muestran en área gris). (b) Un nodo X es independiente condicionalmente de todos los demás nodos en la red dado su manto de Markov (el área gris).

[2] Hay también un criterio topológico general llamado **d-separación** para decidir si un conjunto de nodos **X** es independiente de otro conjunto **Y**, dado un tercer conjunto **Z**. El criterio es más complicado y no es necesario para deducir los algoritmos de este capítulo, así que lo omitimos. Los detalles pueden encontrarse en Russell y Norving (1995) o Pearl (1988). Shachter (1998) da un método más intuitivo para determinar la d-separación.

14.3 Representación eficiente de las distribuciones condicionales

Incluso si el número máximo de padres k es más bien pequeño, rellenar la TPC de un nodo requiere hasta $O(2^k)$ números y quizás una gran cantidad de experiencia con todos los casos de condicionamiento posibles. En realidad, éste es un escenario del peor caso en el que las relaciones entre los padres y los hijos son completamente arbitrarias. Usualmente, tales relaciones están descritas por una **distribución canónica** que corresponde a algún modelo estándar. En tales casos, la tabla completa puede especificarse llamándola como el modelo y quizás aportando unos pocos parámetros, mucho más fácil que proporcionando un número exponencial de parámetros.

DISTRIBUCIÓN CANÓNICA

NODOS DETERMINISTAS

El ejemplo más sencillo lo proporcionan los **nodos deterministas**. Un nodo determinista tiene su valor exactamente especificado por los valores de sus padres, sin incertidumbre. La relación puede ser lógica: por ejemplo, la relación entre los nodos padres *Canadienses*, *EstadoUnidense*, *Mejicano* y el nodo hijo *NorteAmericano* es simplemente que el hijo es la disyunción de los padres. La relación también puede ser numérica: por ejemplo, si los nodos padres son los precios de un modelo particular de coche en distintas tiendas de compra-venta, y el nodo hijo es el precio que termina pagando un cazador de gangas, entonces el nodo hijo es el mínimo de los valores padre; o si los nodos padres son las entradas de un lago (ríos, residuos, precipitaciones) y las salidas del lago (ríos, evaporación, filtraciones) y el hijo es el cambio del nivel de agua del lago, entonces el valor del hijo es la diferencia entre los padres referentes a las entradas y los padres que se refieran a las salidas.

RUIDO-OR

Las relaciones de incertidumbre a menudo pueden caracterizarse por las llamadas relaciones lógicas «ruido». El ejemplo estándar es la relación **ruido-OR**, que es una generalización del OR lógico. En lógica proposicional, diríamos que *Fiebre* es cierto si y sólo si *Resfriado*, *Gripe* o *Malaria* es cierta. El modelo ruido-OR tiene en cuenta la incertidumbre concerniente a la capacidad de cada padre para causar que el hijo sea cierto (la relación causal entre padre e hijo puede ser *inhibida*, y así un paciente podría tener un resfriado, pero no manifestar fiebre). El modelo considera dos suposiciones. Primero, asume que todas las causas posibles están incluidas. (Esto no es tan estricto como parece, ya que siempre podemos añadir el llamado **nodo de filtración** que agrupa a las «causas de todo tipo».) Segundo, asume que la inhibición de cada padre es independiente de la inhibición de cualquier otro padre: por ejemplo, el que se inhiba *Malaria* como causa de la fiebre es independiente de que se inhiba *Gripe* como causa de la fiebre. Dados estos supuestos, *Fiebre* es *falso* si y sólo si todos sus padres *ciertos* están inhibidos, y la probabilidad de ésta es el producto de las probabilidades de inhibición de cada padre. Supongamos que estas probabilidades de inhibición son las siguientes:

NODO DE FILTRACIÓN

$$P(\neg fiebre | resfriado, \neg gripe, \neg malaria) = 0{,}6$$
$$P(\neg fiebre | \neg resfriado, gripe, \neg malaria) = 0{,}2$$
$$P(\neg fiebre | \neg resfriado, \neg gripe, malaria) = 0{,}1$$

Entonces, a partir de esta información y las suposiciones del ruido-OR, puede construirse la TPC entera. La siguiente tabla muestra cómo:

Resfriado	*Gripe*	*Malaria*	*P(Fiebre)*	*P(¬Fiebre)*
F	F	F	0,0	1,0
F	F	V	0,9	**0,1**
F	V	F	0,8	**0,2**
F	V	V	0,98	0,02 = 0,2 × 0,1
V	F	F	0,4	**0,6**
V	F	V	0,94	0,06 = 0,6 × 0,1
V	V	F	0,88	0,12 = 0,6 × 0,2
V	V	V	0,988	0,012 = 0,6 × 0,2 × 0,1

En general, las relaciones lógicas de ruido en las que una variable depende de k padres pueden describirse utilizando $O(k)$ parámetros en vez de $O(2^k)$ para la tabla de probabilidad condicional completa. Estos hace el cálculo y el aprendizaje mucho más fácil. Por ejemplo, la red CPCS (Pradhan *et al.*, 1994) usa distribuciones ruido-OR y ruido-MAX para modelar las relaciones entre enfermedades y síntomas en medicina interna. Con 448 nodos y 906 enlaces, necesita sólo 8.254 valores en vez de 133.931.430 para una red con TPCs completas.

Redes bayesianas con variables continuas

Muchos problemas del mundo real involucran cantidades continuas, tales como altura, masa, temperatura y dinero; de hecho, la mayor parte de la estadística trata con variables aleatorias cuyos dominios son continuos. Por definición, las variables continuas tienen un número infinito de valores posibles, así que es imposible especificar explícitamente las probabilidades condicionadas para cada valor. Una posible forma para manipular variables continuas es obviarlas utilizando la **discretización** (esto es, dividir los valores posibles en un conjunto fijo de intervalos). Por ejemplo, las temperaturas podrían dividirse en (<0°C), (0°C–100°C), y (>100°C). La discretización a veces es una solución aceptable, pero a menudo se traduce en una pérdida considerable de precisión y TPC muy grandes. La otra solución es definir familias estándares de funciones de densidad de probabilidad (*véase* Apéndice A) que estén especificadas por un número finito de **parámetros**. Por ejemplo, una distribución gaussiana (o normal) $N(\mu, \sigma^2)(x)$ tiene la media μ y la varianza σ^2 como parámetros.

DISCRETIZACIÓN

PARÁMETROS

RED BAYESIANA HÍBRIDA

Una red con variables tanto discretas como continuas se llama **red bayesiana híbrida**. Para especificar una red híbrida, tenemos que concretar dos tipos nuevos de distribuciones: la distribución condicionada para una variable continua dados los padres discretos o continuos; y la distribución condicionada para una variable discreta dados los padres continuos. Considere el ejemplo sencillo de la Figura 14.5, en el que un cliente compra fruta dependiendo de su costo, el cual depende a su vez del tamaño de la cosecha y de la aplicación del plan de subvención estatal. La variable *Costo* es continua y tiene padres continuos y discretos; la variable *Compras* es discreta y tiene un padre continuo.

Figura 14.5 Una red sencilla con variables discretas (*Subvención* y *Compras*) y variables continuas (*Cosecha* y *Costo*).

Para la variable *Costo*, necesitamos especificar **P**(*Costo*|*Cosecha, Subvención*). El padre discreto se maneja mediante enumeración explícita, esto es, especificando tanto *P*(*Costo*|*Cosecha, subvención*) como *P*(*Costo*|*Cosecha,* ¬*subvención*). Para manejar *Cosecha*, especificamos cómo la distribución sobre el costo c depende del valor continuo h de *Cosecha*. En otras palabras, especificamos los *parámetro* de la distribución costo como una función de h. La elección más común es la distribución **Gaussiana lineal**, en la que el hijo tiene una distribución Gaussiana cuya media μ varía linealmente con el valor de los padres y cuya desviación estándar σ está fijada. Necesitamos dos distribuciones, una para *subvención* y otra para ¬*subvención*, con parámetros diferentes:

GAUSSIANA LINEAL

$$P(c|h, subvención) = N(a_t h + b_t, \sigma_t^2)(c) = \frac{1}{\sigma_t\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_t h + b_t)}{\sigma_t}\right)^2}$$

$$P(c|h, \neg subvención) = N(a_f h + b_f, \sigma_f^2)(c) = \frac{1}{\sigma_f\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_f h + b_f)}{\sigma_f}\right)^2}$$

Para este ejemplo, entonces, la distribución condicionada para *Costo* está especificada llamándola como distribución gaussiana lineal y aportando los parámetros a_t, b_t, σ_t, a_f, b_f, y σ_f. Las Figuras 14.6(a) y (b) muestran estas dos relaciones. Nótese que en cada caso

Figura 14.6 Las gráficas de (a) y (b) muestran la distribución de probabilidad de *Costo* como una función del tamaño de la *Cosecha*, con *Subvención* cierto y falso respectivamente. La gráfica (c) muestra la distribución *P*(*Costo*|*Cosecha*), obtenida como un sumatorio sobre los dos casos de subvención.

la pendiente es negativa, ya que el precio decrece cuando las existencias crecen. (Por supuesto, el supuesto de linealidad implica que el precio llega a ser negativo en algún punto; el modelo lineal es razonable sólo si el tamaño de la cosecha está limitado a un rango limitado.) La Figura 14.6(c) muestra la distribución $P(c|h)$, promediando en los dos valores posibles de *Subvención* y asumiendo que cada uno tiene una probabilidad *a priori* de 0,5. Esto muestra que incluso con modelos muy sencillos pueden representarse bastantes distribuciones interesantes.

La distribución condicionada Gaussiana lineal tiene algunas propiedades especiales. Una red que contenga sólo variables continuas con distribuciones Gaussianas lineales tiene una distribución conjunta que es una distribución Gaussiana multivariable sobre todas las variables (Ejercicio 14.5)[3]. (Una distribución Gaussiana multivariable es una superficie con más de una dimensión que tiene un pico en la media (de dimensión n) y disminuye por todos lados.) Cuando se añaden las variables discretas (siempre y cuando la variable discreta no sea hija de una variable continua), la red define una distribución **Gaussiana condicionada**, o GC: dada cualquier asignación de las variables discretas, la distribución sobre las variables continuas es una Gaussiana multivariable.

GAUSSIANA CONDICIONADA

Ahora volvemos a las distribuciones de variables discretas con padres continuos. Considere, por ejemplo, el nodo *Compras* de la Figura 14.5. Parece razonable asumir que el cliente comprará si el costo es bajo y no comprará si es alto y que la probabilidad de la compra varía suavemente en algunas regiones intermedias. En otras palabras, la distribución condicionada es como una función umbral «suave». Un modo de construir umbrales suaves es utilizar la *integral* de la distribución normal estándar:

$$\Phi(x) = \int_{-\infty}^{x} N(0, 1)(x)\,dx$$

Entonces la probabilidad de *Compras* dado *Coste* debe ser

$$P(compras|Costo = c) = \Phi((-c + \mu)/\sigma)$$

lo que significa que el umbral de costo se produce alrededor de μ, el ancho de la región del umbral es proporcional a σ, y la probabilidad de la compra decrece cuando el costo crece.

DISTRIBUCIÓN PROBIT

Esta **distribución probit**[4] se ilustra en la Figura 14.7(a). El procedimiento puede justificarse haciendo el supuesto de que el proceso de decisión subyacente tiene un umbral abrupto, pero que la localización exacta del umbral está sometido a ruido Gaussiano aleatorio. Una alternativa al modelo probit es la **distribución logit**,[5] que usa la **función sigmoide** para generar un umbral suave:

DISTRIBUCIÓN LOGIT

FUNCIÓN SIGMOIDE

---

[3] Se deduce que la inferencia en redes Gaussianas lineales necesitan sólo tiempo $O(n^3)$ en el peor caso, sea cual sea la topología de la red. En la Sección 14.4, veremos que la inferencia para redes con variables discretas es NP-duro.

[4] Charle Ittener Bliss (1899-1979) introdujo el modelo y el término probit para simplificar la expresión «probability unit» (unidad de probabilidad). Bliss buscaba en la probit una escala convincente de distribuciones normales (desviadas) que representaran el modelo clásico de los ensayos biológicos (donde los estímulos están determinados y las respuestas son aleatorias debido a la variabilidad de los niveles de estímulo de los individuos). *(Nota del traductor.)*

[5] Joseph Berkson (1899-1982) introdujo en 1944 el término logit, en analogía al término probit de Bliss, al investigar métodos estadísticos en ensayos biológicos. Propuso utilizar la inversa de la función logística ($P(Z)=exp\{Z\}/1+exp\{Z\}$) en vez de funciones de probabilidad normales. *(Nota del traductor.)*

Figura 14.7 (a) Una distribución probit para la probabilidad de *Compras* dado *Costo*, con $\mu = 6{,}0$ y $\sigma = 1{,}0$. (b) Una distribución logit con los mismos parámetros.

$$P(compras|Costo = c) = \frac{1}{1 + exp\left(-2\,\frac{-c + \mu}{\sigma}\right)}$$

Esto se ilustra en la Figura 14.7(b). Las dos distribuciones tienen una apariencia similar, pero la logit tiene unas «colas» mucho más largas. La probit es a menudo un ajuste mejor para situaciones reales, pero la logit es a veces más fácil de tratar matemáticamente. Es ampliamente utilizada en redes neuronales (Capítulo 20). Tanto la probit como la logit pueden generalizarse para manejar múltiples padres continuos, tomando una combinación lineal de los valores de los padres. En el Ejercicio 14.6 se investigan extensiones para un hijo discreto multivaluado.

14.4 Inferencia exacta en redes bayesianas

La tarea básica de cualquier sistema de inferencia probabilista es computar la distribución de probabilidad *a posteriori* para un conjunto de **variables pregunta**, dado algún **evento** observado (esto es, alguna asignación de valores para un conjunto de **variables evidencia**). Usaremos la notación introducida en el Capítulo 13: X denota a la variable pregunta; $\mathbf{E}$ denota al conjunto de variables evidencias $E_1, \ldots, E_m$, y $\mathbf{e}$ es un evento observado particular; $\mathbf{Y}$ denotará las variables no evidencia $Y_1, \ldots, Y_l$ (a veces llamadas **variables ocultas**). Así, el conjunto completo de variables es $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$. Una cuestión típica pide la distribución de probabilidad *a posteriori* $\mathbf{P}(X|\mathbf{e})$[6].

EVENTO

VARIABLES OCULTAS

[6] Asumiremos que la variable pregunta no está entre las variables evidencia; de serlo, entonces la distribución posteriori de X simplemente dará 1 al valor observado. Por simplicidad, también hemos asumido que la pregunta es exactamente una sola variable. Nuestro algoritmo puede extenderse fácilmente para manejar preguntas conjuntas sobre varias variables.

En la red del robo, podemos observar el evento en el que *JohnLlama=cierto* y *MaryLlama=cierto*. Podríamos entonces preguntarnos por, digamos, la probabilidad de que haya ocurrido un robo:

$$\mathbf{P}(Robo|JohnLlama = cierto, MaryLlama = cierto)=\langle 0{,}284,\ 0{,}716\rangle$$

En esta sección discutiremos los algoritmos exactos para la computación de probabilidades posteriori y estimaremos la complejidad de esta tarea. Resulta que el caso general es intratable, así que la Sección 14.5 trata métodos para inferencia aproximada.

Inferencia por enumeración

El Capítulo 13 explicó que cualquier probabilidad condicionada puede computarse sumando términos de la distribución conjunta completa. Más específicamente, una pregunta $\mathbf{P}(X|\mathbf{e})$ puede responderse utilizando la Ecuación (13.6), que repetimos aquí por conveniencia:

$$\mathbf{P}(X|\mathbf{e}) = \alpha\, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$$

Ahora, una red bayesiana proporciona una representación completa de la distribución conjunta completa. Más concretamente, la Ecuación (14.1) muestra que los términos $P(x, \mathbf{e}, \mathbf{y})$ de la distribución conjunta pueden escribirse como productos de probabilidades condicionadas de la red. Así, *una pregunta puede responderse utilizando una red bayesiana computando sumas de productos de probabilidades condicionadas de la red*.

En la Figura 13.4, un algoritmo, PREGUNTAR-CONTAR-CONJUNTA, se propuso para inferencia por enumeración a partir de la distribución conjunta completa. El algoritmo toma como entrada una distribución conjunta completa **P** y busca allí valores. Es un problema sencillo modificar el algoritmo para que tome como entrada una red bayesiana *rb* y «busque» entradas conjuntas multiplicando los correspondientes datos de las TPCs de la *rb*.

Considere la pregunta $\mathbf{P}(Robo|JohnLlama = cierto, MaryLlama = cierto)$. Las variables ocultas para esta pregunta son *Terremoto* y *Alarma*. De la Ecuación (13.6), utilizando las letras iniciales de las variables para acortar las expresiones, tenemos[7]

$$\mathbf{P}(B|j, m) = \alpha\, \mathbf{P}(B, j, m) = \alpha \sum_{e} \sum_{a} \mathbf{P}(B, e, a, j, m)$$

La semántica de las redes bayesianas (Ecuación 14.1) nos dio entonces una expresión en términos de las entradas de la TPC. Por simplicidad, haremos esto sólo para *Robo=cierto*:

$$\mathbf{P}(b|j, m) = \alpha \sum_{e} \sum_{a} P(b)P(e)P(a|b, e)P(j|a)P(m|a)$$

Para computar esta expresión, tenemos que sumar cuatro términos, cada uno calculado como la multiplicación de cinco números. En el peor caso, cuando tengamos que sumar

[7] Una expresión tal como $\sum_e P(a, e)$ significa sumar $P(A=a,E=e)$ para todos los posibles valores de e. Hay una ambigüedad y es que $P(e)$ se utiliza para indicar tanto $P(E=cierto)$ y $P(E=e)$, pero el significado que se pretende debería estar claro del contexto; en particular, en el contexto de la suma se pretende el último significado.

casi todas las variables, la complejidad del algoritmo para una red con n variables booleanas es $O(n2^n)$.

Puede obtenerse una mejora considerando la siguiente observación simple: el término $P(b)$ es una constante y puede moverse fuera de las sumatorias en a y e, y el término $P(e)$ puede moverse fuera de la sumatoria en a. Así, tenemos:

$$\mathbf{P}(b|j, m) = \alpha\, P(b) \sum_e P(e) \sum_a P(a|b, e)P(j|a)P(m|a) \tag{14.3}$$

Esta expresión puede evaluarse ciclando a través de las variables por orden, multiplicando las entradas de la TPC sobre la marcha. Para cada sumatoria, también necesitamos ciclar en los valores posibles de las variables. La estructura de esta computación se muestra en la Figura 14.8. Utilizando los números de la Figura 14.2, obtenemos $P(b|j,m) = \alpha \times 0{,}00059224$. El cálculo correspondiente para $\neg b$ produce $\alpha \times 0{,}0014919$; así:

$$\mathbf{P}(B|j, m) = \alpha\langle 0{,}00059224, 0{,}0014919\rangle \approx \langle 0{,}284, 0{,}716\rangle$$

Esto es, el riesgo de un robo, dadas las llamadas de los dos vecinos, está alrededor del 28 por ciento.

El proceso de evaluación para la expresión de la Ecuación (14.3) se muestra como una expresión arbórea en la Figura 14.8. El algoritmo PREGUNTAR-POR-ENUMERACIÓN de la Figura 14.9 evalúa esos árboles usando la exploración recursiva de primero-en-profundidad. De este modo, la complejidad del espacio de PREGUNTAR-POR-ENUMERACIÓN es sólo lineal en el número de variables (efectivamente, el algoritmo suma con la distribución conjunta completa sin jamás construirla explícitamente). Desafortunadamente, su complejidad en tiempo para una red con n variables booleanas es siempre $O(2^n)$, preferible al $O(n2^n)$ del método sencillo descrito anteriormente, pero todavía bastante desastroso. Una cosa a apuntar sobre el árbol de la Figura 14.8 es que construye explícitamente las *subexpresiones repetidas* que son evaluadas por el algoritmo. Los productos $\mathbf{P}(j|a)\mathbf{P}(m|a)$ y

Figura 14.8 La estructura de la expresión que se indica en la Ecuación (14.3). La evaluación procede de arriba-abajo, multiplicando los valores que están a lo largo de cada camino y sumando en los nodos «+». Nótese la repetición de los caminos para j y m.

```
función PREGUNTAR-POR-ENUMERACION(X, e, rb) devuelve una distribución sobre X
   entradas: X, la variable pregunta
             e, los valores observados de las variables E
             rb, una red bayesiana con variables {X} ∪ E ∪ Y   /* Y = variables ocultas */

   Q(X) ← una distribución sobre X, inicialmente está vacía
   para cada valor x_i de X hacer
      ampliar e con el valor x_i de X
      Q(x_i) ← ENUMERAR-TODO(VARS[rb], e)
   devolver NORMALIZAR (Q(X))
```

```
función ENUMERAR-TODO (vars, e) devuelve un número real
   si VACIO?(vars) entonces devuelve 1,0
   Y ← PRIMERO(vars)
   si Y tiene valor y en e
      entonces devolver P(y|padres(Y)) × CONTAR-TODO(RESTO(vars), e)
      si no devolver Σ_y P(y|padres(Y)) × CONTAR-TODO(RESTO(vars), e_y)
         donde e_y es igual a e ampliado con Y = y
```

Figura 14.9 El algoritmo de enumeración para responder a preguntas en redes bayesianas.

$\mathbf{P}(j|\neg a)\mathbf{P}(m|\neg a)$ son calculados dos veces, una vez para cada valor de e. La siguiente sección describe un método general que evita esos cálculos desaprovechados.

El algoritmo de eliminación de variables

El algoritmo de enumeración puede mejorarse sustancialmente eliminando los cálculos repetidos del tipo que se ilustraron en la Figura 14.8. La idea es sencilla: hacer los cálculos una sola vez y salvar el resultado para usarlo más tarde. Esto es un tipo de programación dinámica. Hay distintas versiones para esta aproximación; presentamos el algoritmo de **eliminación de variables**, que es el más sencillo. La eliminación de variables funciona evaluando expresiones como la de la Ecuación (14.3) en un orden de *derecha a izquierda* (es decir, *arriba-abajo* en la Figura 14.8). Los resultados intermedios se almacenan, y las sumatorias en cada variable se hacen sólo para aquellas porciones de la expresión que dependen de la variable.

ELIMINACIÓN DE VARIABLES

Ilustremos este proceso para la red del robo. Evaluamos la expresión

$$\mathbf{P}(B|j, m) = \alpha \underbrace{\mathbf{P}(B)}_{B} \sum_e \underbrace{P(e)}_{E} \sum_a \underbrace{\mathbf{P}(a|B, e)}_{A} \underbrace{P(j|a)}_{J} \underbrace{P(m|a)}_{M}$$

Nótese que hemos comentado cada parte de la expresión con el nombre de la variable asociada; estas partes se llaman **factores**. Los pasos son como sigue:

FACTORES

- El factor para M, $P(m|a)$, no requiere sumar en M (ya que el valor de M está ya fijado). Almacenamos la probabilidad, dado cada valor de a, en un vector de dos elementos,

$$\mathbf{f}_M(A) = \begin{pmatrix} P(m|a) \\ P(m|\neg a) \end{pmatrix}$$

  ($\mathbf{f}_M$ significa que M fue utilizado para obtener $\mathbf{f}$.)

- Análogamente, almacenamos el factor para J como el vector de dos elemento $\mathbf{f}_J(A)$.
- El factor para A es $\mathbf{P}(a|B, e)$, que será una matriz $\mathbf{f}_J(A, B, E)$ de dimensión $2 \times 2 \times 2$.
- Ahora debemos sumar en A el producto de estos tres factores. Esto nos dará una matriz 2×2 cuyos índices comprenden sólo a B y a E. Ponemos una barra sobre A en el nombre de la matriz para indicar que se ha sumado en A:

$$\begin{aligned}\mathbf{f}_{\bar{A}JM}(B, e) &= \sum_a \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a)\\ &= \mathbf{f}_A(a, B, E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a)\\ &\quad + \mathbf{f}_A(\neg a, B, E) \times \mathbf{f}_J(\neg a) \times \mathbf{f}_M(\neg a)\end{aligned}$$

PRODUCTO PUNTO A PUNTO

El proceso de multiplicación utilizado aquí se llama **producto punto a punto** y se explicará dentro de poco.

- Procesamos E del mismo modo: sumamos en E el producto de $\mathbf{f}_E(E)$ y $\mathbf{f}_{\bar{A}JM}(B, E)$:

$$\begin{aligned}\mathbf{f}_{\bar{E}\bar{A}JM}(B) &= \mathbf{f}_E(e) \times \mathbf{f}_{\bar{A}JM}(B, e)\\ &\quad + \mathbf{f}_E(\neg e) \times \mathbf{f}_{\bar{A}JM}(B, \neg e)\end{aligned}$$

- Ahora podemos calcular la respuesta simplemente multiplicando el factor para B (e.d., $\mathbf{f}_B(B) = \boldsymbol{P}(B)$) por la matriz acumulada $\mathbf{f}_{\bar{E}\bar{A}JM}(B)$:

$$\mathbf{P}(B|j, m) = \alpha\, \mathbf{f}_B(B) \times \mathbf{f}_{\bar{E}\bar{A}JM}(B)$$

El Ejercicio 14.7(a) le pide que compruebe que este proceso produce la respuesta correcta.

Examinando esta secuencia de pasos, vemos que hay dos operaciones computacionales necesarias: el producto punto a punto de un par de factores, y la sumatoria en una variable del producto de los factores.

El producto punto a punto no es una multiplicación matricial, ni tampoco la multiplicación elemento por elemento. El producto punto a punto de dos factores $\mathbf{f}_1$ y $\mathbf{f}_2$ produce un nuevo factor $\mathbf{f}$ cuyas variables son la unión de las variables de $\mathbf{f}_1$ y de $\mathbf{f}_2$. Suponga que los dos factores que tienen las variables $Y_1 \ldots, Y_k$ en común. Entonces tenemos

$$\mathbf{f}(X_1 \ldots X_j, Y_1 \ldots Y_k, Z_1 \ldots Z_l) = \mathbf{f}_1(X_1 \ldots X_j, Y_1 \ldots Y_k)\, \mathbf{f}_2(Y_1 \ldots Y_k, Z_1 \ldots Z_l)$$

Si todas las variables son binarias, entonces $\mathbf{f}_1$ y $\mathbf{f}_2$ tienen 2^{j+k} y 2^{k+l} entradas respectivamente, y el producto punto a punto tiene 2^{j+k+l} entradas. Por ejemplo, dados dos factores $\mathbf{f}_1(A,B)$ y $\mathbf{f}_2(B,C)$ con las distribuciones de probabilidad que se muestran abajo, el producto punto a punto $\mathbf{f}_1 \times \mathbf{f}_2$ viene dado por $\mathbf{f}_3(A, B, C)$:

A	B	$\mathbf{f}_1(A,B)$	B	C	$\mathbf{f}_2(B,C)$	A	B	C	$\mathbf{f}_3(A,B,C)$
V	V	.3	V	V	.2	V	V	V	$.3 \times .2$
V	F	.7	V	F	.8	V	V	F	$.3 \times .8$
F	V	.9	F	V	.6	V	F	V	$.7 \times .6$
F	F	.1	F	F	.4	V	F	F	$.7 \times .4$
						F	V	V	$.9 \times .2$
						F	V	F	$.9 \times .8$
						F	F	V	$.1 \times .6$
						F	F	F	$.1 \times .4$

Hacer la sumatoria en una variable a partir del producto de factores es también una computación bastante sencilla. El único truco es darse cuenta de que cualquier factor que *no* dependa de la variable que va a ser sumada puede excluirse de la sumatoria. Por ejemplo,

$$\sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) \times \mathbf{f}_J(A) \times \mathbf{f}_M(A) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e)$$

Ahora se calcula el producto punto a punto que está en la sumatoria, y se suma en la variable a partir de la matriz resultante.

$$\mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A, B, e) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \mathbf{f}_{\bar{E},A}(A, B)$$

Nótese que las matrices *no* se multiplican hasta que necesitemos sumar una variable del producto acumulado. En ese punto, multiplicamos aquellas matrices que incluyan a la variable que va a ser sumada. Dadas las rutinas para el producto punto a punto y para la sumatoria, el algoritmo de eliminación de variables propiamente dicho puede escribirse de forma bastante sencilla, como se muestra en la Figura 14.10.

Consideremos una pregunta más: $P(JohnLlama|Robo=cierto)$. Como siempre, el primer paso es escribir la sumatoria completa como sumatorias anidadas:

$$P(J|b) = \alpha\, P(b) \sum_e P(e) \sum_a \mathbf{P}(a|b, e) P(J|a) \sum_m P(m|a)$$

Si evaluamos esta expresión de derecha a izquierda, nos damos cuenta de una cosa interesante: ¡$\sum_m P(m|a)$ es igual a 1 por definición! Por lo tanto, no hubo necesidad de incluirlo en primer lugar; la variable M es *irrelevante* para esta pregunta. Otro modo de decir esto es que el resultado de la pregunta $P(JohnLlama|Robo=cierto)$ no cambia si quitamos totalmente a *MaryLLama* de la red. En general, podemos quitar cualquier nodo hoja que no es una variable pregunta o una variable evidencia. Después de quitarla, quizás haya algunos nodos hojas más, y éstas también pueden ser irrelevantes. Continuando este proceso, al final encontramos *que cada variable que no es un ancestro de la variable pregunta o variable evidencia es irrelevante para la pregunta*. Un algoritmo de eliminación de variables puede así quitar todas aquellas variables antes de evaluar la pregunta.

```
función PREGUNTAR-POR-ELIMINACIÓN(X, e, rb) devuelve una distribución sobre X
  entradas: X, la variable pregunta
            e, evidencia establecida como un evento
            rb, una red bayesiana que especifique una distribución conjunta P(X1..., Xn)

  factores ← [ ]; vars ← INVERTIR(VARS[rb])
  para cada var en vars hacer
    factores ← [CREAR-FACTOR(var, e)|factores]
    si var es una variable oculta entonces factores ← SUMA(var, factores)
  devolver NORMALIZAR(PRODUCTO-PUNTO-A-PUNTO (factores))
```

Figura 14.10 El algoritmo de eliminación de variables para responder a preguntas en redes bayesianas.

La complejidad de la inferencia exacta

Hemos indicado que la eliminación de variables es más eficiente que la enumeración porque evita cálculos repetidos (aparte de lo de «tirar» las variables irrelevantes). Los requerimientos de tiempo y espacio de la eliminación de variables están dominados por el tamaño del factor más grande que se construya durante el funcionamiento del algoritmo. Éste, a su vez, está determinado por el orden de eliminación de variables y por la estructura de la red.

La red del robo de la Figura 14.2 pertenece a la familia de redes en las que hay a lo sumo un camino no dirigido entre dos nodos cualesquiera de la red. Éstas se llaman redes con **conexión simple** o **poliárboles**, y tienen en particular una buena propiedad: *La complejidad en tiempo y espacio de la inferencia exacta en poliárboles es lineal en el tamaño de la red*. Aquí, el tamaño está definido con el número de entradas de la TPC; si el número de padres de cada nodo está acotado por una constante, entonces la complejidad también será lineal en el número de nodos. Estos resultados se verifican para cualquier ordenación consistente con la ordenación topológica de la red (Ejercicio 14.7).

CONEXIÓN SIMPLE

POLIÁRBOLES

Para redes con **conexión múltiple**, como la de la Figura 14.11(a), la eliminación de variables puede tener complejidad exponencial en tiempo y espacio en el peor de los casos, incluso cuando el número de padres por nodo esté acotado. Esto no es sorprendente cuando uno considera que, *ya que incluye a la inferencia de la lógica proposicional como un caso particular, la inferencia en redes bayesianas es NP-duro*. De hecho, puede demostrarse (Ejercicio 14.8) que el problema es tan duro como el del cálculo del *número* de asignaciones satisfactorias para una fórmula lógica proposicional. Esto significa que es #P-duro («número-P duro»), es decir, estrictamente más duro que los problemas NP-completos.

CONEXIÓN MÚLTIPLE

Hay una conexión directa entre la complejidad de la inferencia de redes bayesianas y la complejidad de los problemas de satisfacción de restricciones (PSRs). Como analizamos en el Capítulo 5, la dificultad de resolver un PSR está relacionado a cómo de «parecido a un árbol» es su grafo de restricciones. Medidas como el **ancho del hiperárbol**, que acota la complejidad de la resolución de un PSR, puede también aplicarse directamente a redes bayesianas. Por otra parte, el algoritmo de eliminación de variables puede generalizarse para resolver PSRs además de las redes bayesianas.

Algoritmos basados en grupos

El algoritmo de eliminación de variables es sencillo y eficiente para responder a preguntas particulares. Si queremos computar probabilidades posteriores para todas las variables de una red puede, sin embargo, ser menos eficiente. Por ejemplo, en una red poliárbol, necesitaría para informar de $O(n)$ preguntas que cuesten $O(n)$ cada una, un tiempo total del orden $O(n^2)$. Utilizando el algoritmo **basado en grupos** (también conocido como algoritmos basados en **árboles de uniones**), el tiempo puede reducirse a $O(n)$. Por esta razón, estos algoritmos tienen un uso muy extendido en herramientas comerciales de redes bayesianas.

GRUPOS

ÁRBOL DE UNIONES

La idea básica del agrupamiento (creación de grupos) es unir nodos particulares de la red para crear nodos de grupos de tal modo que la red resultante sea un poliárbol. Por

Figura 14.11 (a) Una red múltiplemente conectada con las tablas de probabilidad condicional. (b) Un agrupamiento equivalente a la red múltiplemente conectada.

ejemplo, la red con conexión múltiple de la Figura 14.11(a) puede convertirse en un poliárbol uniendo los nodos *Aspersor* y *Lluvia* en un nodo-grupo denominado *Aspersor* + *Lluvia*, como muestra la Figura 14.11(b). Los dos nodos booleanos son sustituidos por un meganodo que toma cuatro posibles valores: *VV*, *VF*, *FV* y *FF*. El meganodo sólo tiene un padre, la variable booleana *Nublado*, por lo que hay dos casos para el condicionamiento.

Una vez que la red está en forma de poliárbol, se aplica un algoritmo de inferencia de propósito específico. Esencialmente, el algoritmo es un tipo de propagación con restricciones (Capítulo 5) donde las restricciones garantizan que los grupos vecinos coinciden en la probabilidad posteriori de cualquier variable que tengan en común. Con un cómputo meticuloso, este algoritmo es capaz de calcular probabilidades posteriores para todos los nodos no-evidencia de la red en tiempo $O(n)$, donde n es ahora el tamaño de la red modificada. Sin embargo, no desaparece el que el problema siga siendo NP-duro: si una red requiere tiempo y espacio exponencial con la eliminación de variables, entonces las TPCs en la red de grupos requerirá tiempo y espacio exponencial para construirlo.

14.5 Inferencia aproximada en redes bayesianas

MONTE CARLO

Dada la intratabilidad de la inferencia exacta en grandes redes con conexión múltiple es imprescindible considerar métodos de inferencia aproximados. Esta sección describe algoritmos de muestreo aleatorio, también llamados algoritmos de **Monte Carlo**, que proporcionan respuestas aproximadas cuya precisión depende del número de muestras generadas. En los últimos años, los algoritmos Monte Carlo han llegado a ser ampliamente utilizados en informática para estimar cantidades que son difíciles de calcular de una forma exacta. Por ejemplo, el algoritmo de enfriamiento simulado descrito en

el Capítulo 4 es un método de Monte Carlo para problemas de optimización. En esta sección, estamos interesados en el muestreo aplicado al cálculo de probabilidades posteriores. Describimos dos familias de algoritmos: el muestreo directo y el muestreo mediante cadenas de Markov. En las notas del final del capítulo se mencionarán otras dos aproximaciones: los métodos variacionales y la propagación cíclica.

Métodos de muestreo directo

El elemento principal en cualquier algoritmo de muestreo es la generación de muestras a partir de una distribución de probabilidad conocida. Por ejemplo, se puede pensar en una moneda no trucada como una variable aleatoria *Moneda* con los valores ⟨*cara,cruz*⟩ y una distribución priori $\mathbf{P}(Moneda) = \langle 0{,}5, 0{,}5\rangle$. Muestrear a partir de esta distribución es exactamente igual al lanzamiento de la moneda: devolverá *cara* con probabilidad 0,5 y con probabilidad 0,5 devolverá *cruz*. Dada una serie de números aleatorios en el intervalo [0,1], muestrear cualquier distribución de una sola variable es un problema sencillo. (*Véase* Ejercicio 14.9.)

La clase más sencilla de procedimientos de muestreo aleatorio para redes bayesianas genera sucesos a partir de una red que no tenga ninguna evidencia asociada. La idea es muestrear cada variable por turno, en orden topológico. La distribución de probabilidad usada para obtener el valor muestreado está condicionada a los valores que ya se han asignado a los padres de dicha variable. Este algoritmo se muestra en la Figura 14.12. Podemos ilustrar su funcionamiento con la red de la Figura 14.11(a), suponiendo el orden [*Nublado, Aspersor, Lluvia, HierbaHúmeda*]:

1. Muestreo a partir de $\mathbf{P}(Nublado) = \langle 0{,}5, 0{,}5\rangle$; suponga que esto devuelve *verdad*.
2. Muestreo a partir de $\mathbf{P}(Aspersor|Nublado = verdad) = \langle 0{,}1, 0{,}9\rangle$; suponga que devuelve *falso*.
3. Muestreo a partir de $\mathbf{P}(Lluvia|Nublado = verdad) = \langle 0{,}2, 0{,}8\rangle$; suponga que devuelve *verdad*.
4. Muestreo a partir de $\mathbf{P}(HierbaHumeda \mid Aspersor = falso, Lluvia = verdad) = \langle 0{,}9, 0{,}1\rangle$; suponga que devuelve *verdad*.

En este caso, el MUESTREO-POR-PRIORI devuelve el suceso [*verdad, falso, verdad, verdad*].

Es fácil ver que el MUESTREO-POR-PRIORI genera muestras a partir de la distribución conjunta *a priori* determinada por la red. Primero, sea $S_{MP}(x_1 \ldots, x_n)$ la probabilidad de que un suceso particular esté generado por el algoritmo MUESTREO-POR-PRIORI. Sólo con mirar el proceso de muestreo, tenemos que

$$S_{MP}(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i | padres(X_i))$$

ya que cada paso del muestreo depende sólo de los valores de los padres. Esta expresión debería resultar familiar, puesto que es también la probabilidad del suceso según la representación de la distribución conjunta de la red bayesiana, como se indica en la Ecuación (14.1). Esto es, tenemos que

$$S_{MP}(x_1 \ldots x_n) = P(x_1 \ldots x_n)$$

Este simple resultado hace muy fácil responder a preguntas utilizando muestras.

función MUESTREO-POR-PRIORI(*rb*) **devuelve** un suceso muestreado a partir de una *a priori* especificada por *rb*.
  entradas: *rb*, una red bayesiana especificando la distribución conjunta $\mathbf{P}(X_1 \ldots, X_n)$

  x ← un suceso con *n* elementos
  desde $i = 1$ **hasta** *n* **hacer**
    x_i ← una muestra aleatoria de $\mathbf{P}(X_i | padres(X_i))$
  devolver x

Figura 14.12 Algoritmo de muestreo que genera sucesos a partir de una red bayesiana.

En cualquier algoritmo de muestreo, las respuestas se calculan contando las muestras que se hayan generado. Suponga que hay N muestras en total, y sea $N(x_1 ..., x_n)$ la frecuencia del suceso particular $x_1 ..., x_n$. Esperamos que esta frecuencia converja, en el límite, a su valor esperado de acuerdo con la probabilidad de muestreo:

$$\lim_{N \to \infty} \frac{N_{MP}(x_1 \ldots, x_n)}{N} = S_{MP}(x_1 \ldots, x_n) = P(x_1 \ldots, x_n) \tag{14.4}$$

Por ejemplo, considere el suceso producido anteriormente: [*verdad, falso, verdad, verdad*]. La probabilidad de muestreo para este suceso es:

$$S_{MP}(verdad, falso, verdad, verdad) = 0{,}5 \times 0{,}9 \times 0{,}8 \times 0{,}9 = 0{,}324.$$

Así, en el límite de N grandes, esperamos que el 32,4 por ciento de las muestras sean este suceso.

En lo que sigue, cada vez que utilicemos una igualdad aproximada («≈») lo interpretaremos exactamente en este sentido (que la probabilidad estimada se convierta en exacta en el límite de grandes muestras). Tal estimación se llama **consistente**. Por ejemplo, se puede producir una estimación consistente de la probabilidad de cualquier suceso que se especifique parcialmente $x_1 \ldots, x_m$ con $m \leq n$, como sigue:

CONSISTENTE

$$P(x_1 \ldots, x_m) \approx N_{MP}(x_1 \ldots, x_n)/N \tag{14.5}$$

Esto es, la probabilidad del suceso puede estimarse como la fracción de todos los sucesos completos generados por el proceso de muestreo que coinciden con el suceso que se especificó parcialmente. Por ejemplo, si generamos 1.000 muestras a partir de la red del aspersor, y 511 de ellas tienen *Lluvia = verdad*, entonces la probabilidad estimada de llover, que se escribe como $\hat{P}(Lluvia = verdad)$, es 0,511.

Muestreo por rechazo en redes bayesianas

MUESTREO POR RECHAZO

El **muestreo por rechazo** es un método general para generar muestras a partir de una distribución difícil de muestrear utilizando una distribución que sea fácil de muestrear. En su forma más simple, puede usarse para calcular probabilidades condicionadas, esto es, para determinar $P(X|\mathbf{e})$. El algoritmo del MUESTREO-POR-RECHAZO se muestra en la Figura 14.13. Primero, genera muestras a partir de la distribución priori determinada por la red. Después, rechaza todas aquellas que no se correspondan con

```
función MUESTREO-POR-RECHAZO(X, e, rb, N) devuelve una estimación de P(X|e)
  entradas: X, la variable pregunta
            e, evidencia establecida como un evento
            rb, una red bayesiana
            N, el número total de muestras a generar
  variables locales: N, un vector contador sobre X, inicialmente a cero

  desde j = 1 hasta N hacer
    x ← MUESTREO-POR-PRIORI(rb)
    si x es consistente con e entonces
    N[x] ← N[x] + 1 donde x es el valor de X en x
  devolver NORMALIZAR(N[X])
```

Figura 14.13 El algoritmo de muestreo por rechazo para responder preguntas con evidencia en una red bayesiana.

la evidencia. Finalmente, la estimación $\hat{P}(X = x|\mathbf{e})$ se obtiene contando la frecuencia con que ocurre $X = x$ en las muestras que queden.

Sea $\hat{\mathbf{P}}(X|\mathbf{e})$ la distribución estimada que devuelve el algoritmo. A partir de la definición del algoritmo, tenemos que:

$$\hat{\mathbf{P}}(X|\mathbf{e}) = \alpha\, \mathbf{N}_{MP}(X, \mathbf{e}) = \frac{\mathbf{N}_{MP}(X, \mathbf{e})}{\mathbf{N}_{MP}(\mathbf{e})}$$

De la Ecuación (14.5), ésta se convierte en

$$\hat{\mathbf{P}}(X|\mathbf{e}) \approx \frac{\mathbf{P}(X, \mathbf{e})}{\mathbf{P}(\mathbf{e})} = \mathbf{P}(X|\mathbf{e})$$

Esto es, el muestreo por rechazo genera una estimación consistente de la auténtica probabilidad.

Continuando con nuestro ejemplo de la Figura 14.11(a), supongamos que queremos estimar $\mathbf{P}(Lluvia|Aspersor = verdad)$, utilizando 100 muestras. De las 100 que generamos, suponga que 73 tienen $Aspersor = falso$ y son rechazadas, mientras que 27 tienen $Aspersor = verdad$; de las 27, ocho tienen $Lluvia = verdad$ y 19 tienen $Lluvia = falso$. Por lo tanto,

$$\mathbf{P}(Lluvia|Aspersor = verdad) \approx \text{NORMALIZAR}(\langle 8, 19\rangle) = \langle 0{,}296, 0{,}704\rangle$$

La respuesta verdadera es $\langle 0{,}3, 0{,}7\rangle$. Cuantas más muestras se acumulen, la estimación convergerá a la respuesta verdadera. La desviación estándar del error de cada probabilidad será proporcional a $1/\sqrt{n}$, donde n es el número de muestras utilizadas en la estimación.

El mayor problema con el muestreo por rechazo es que ¡rechaza demasiadas muestras! El porcentaje de muestras consistentes con la evidencia **e** desciende exponencialmente conforme crece el número de variables de evidencia, por lo que el procedimiento es simplemente inutilizable para problemas complejos.

Nótese que el muestreo por rechazo es muy similar a la estimación directa de probabilidades condicionadas a partir del mundo real. Por ejemplo, para estimar $\mathbf{P}(Lluvia|Cielo-$

RojoDeNoche = *verdad*), simplemente puede contar con qué frecuencia llueve después de haber observado un cielo rojo la noche anterior, ignorando aquellas noches en las que el cielo no es rojo. (Aquí, el mundo mismo interpreta el papel del algoritmo de generación de muestras.) Obviamente, esto podría tardar mucho tiempo si rara vez el cielo está rojo y esa es la debilidad del muestreo por rechazo.

Ponderación de la verosimilitud

PONDERACIÓN DE LA VEROSIMILITUD

La **ponderación de la verosimilitud** evita la ineficiencia del muestreo de rechazo generando únicamente sucesos que sean consistentes con la evidencia **e**. Comenzamos con la descripción de cómo funciona el algoritmo; después mostraremos que funciona correctamente, esto es, que genera estimaciones de probabilidad consistentes.

PONDERACIÓN-VEROSIMILITUD (*véase* Figura 14.14) fija los valores para las variables de evidencia **E** y sólo muestrea las variables restantes, X e **Y**. Esto garantiza que cada suceso generado es consistente con la evidencia. Sin embargo, no todos los sucesos son iguales. Antes de ajustar la suma en la distribución de la variable pregunta, cada suceso se pondera por la *verosimilitud* de que el suceso concuerde con la evidencia, que se calcula exactamente como el producto de las probabilidades condicionadas de cada variable evidencia (dados sus padres). Intuitivamente, a los sucesos en los que la evidencia presente los hace menos creíbles se les deberían dar menos peso.

Apliquemos el algoritmo sobre la red que se muestra en la Figura 14.11(a), con la pregunta $\mathbf{P}(Lluvia|Aspersor = verdad, HierbaHúmeda = verdad)$. El procedimiento funciona como sigue: primero, el peso de w se establece a 1,0. Después se genera un suceso:

1. Muestreo a partir de $\mathbf{P}(Nublado) = \langle 0{,}5, 0{,}5\rangle$; suponga que esto devuelve *verdad*.
2. *Aspersor* es una variable de evidencia con valor *verdad*. Por lo tanto, establecemos que

$$w \leftarrow w \times P(Aspersor = verdad|Nublado = verdad) = 0{,}1$$

3. Muestreo a partir de $\mathbf{P}(Lluvia|Nublado = verdad) = \langle 0{,}8, 0{,}2\rangle$; suponga que devuelve *verdad*.
4. *HierbaHúmeda* es una variable de evidencia con valor *verdad*. Por lo tanto, establecemos que

$$w \leftarrow w \times P(HierbaHúmeda = verdad|Aspersor = verdad, Lluvia = verdad) = 0{,}099$$

Aquí MUESTRA-PONDERADA devuelve el suceso [*verdad, verdad, verdad, verdad*] con peso 0,099, y esto se ajusta con *Lluvia* = *verdad*. El peso es pequeño porque el suceso describe un día nublado, lo que hace menos creíble que el aspersor esté encendido.

Para entender por qué la ponderación de la verosimilitud funciona, empezamos examinando la distribución de muestreo S_{MW} para MUESTREO-PONDERADO[8]. Recuerde que las variables de evidencia **E** están fijadas a los valores **e**. Llamaremos a las demás va-

[8] Si bien lo correcto sería indicar dicha función como S_{MP} para referirnos a *M*uestreo *P*onderado, se utilizará la notación S_{MW}, pues la notación S_{MP} ya se utilizó para referirnos al Muestreo por Priori (*véase* Ecuación 14.4). *(Nota del traductor.)*

```
función PONDERACIÓN-VEROSIMILITUD(X, e, rb, N) devuelve una estimación de P(X|e)
  entradas: X, la variable pregunta
            e, evidencia establecida como un evento
            rb, una red bayesiana
            N, el número total de muestras a generar
  variables locales: W, un vector de contadores (pesos) para X, inicialmente nulos.

  desde j = 1 hasta N hacer
    x, w ← MUESTRA-PONDERADA(rb)
    W[x] ← W[x] + w donde x es el valor de X en x
  devolver NORMALIZAR(W[X])
```

```
función MUESTRA-PONDERADA(rb, e) devuelve un suceso y un peso

  x ← un suceso con n elementos; w ← 1
  desde i = 1 hasta n hacer
    si X_i tiene un valor x_i en e
      entonces w ← w × P(X_i = x_i | padres(X_i))
      si no x_i ← una muestra aleatoria de P(X_i | padres(X_i))
  devolver x, w
```

Figura 14.14 El algoritmo de ponderación de la verosimilitud para inferencia en redes bayesianas.

riables $\mathbf{Z}$, esto es, $\mathbf{Z} = \{X\} \cup \mathbf{Y}$. El algoritmo obtiene una muestra de cada variable en $\mathbf{Z}$ dados los valores de su padre:

$$S_{MP}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | padres(Z_i)) \tag{14.6}$$

Nótese que $Padres(Z_i)$ pueden incluir tanto variables ocultas como variables de evidencia. A diferencia de la distribución *a priori* $P(\mathbf{z})$, la distribución S_{MW} presta alguna atención a la evidencia: los valores muestreados para cada variable Z_i estarán influenciados por la evidencia que esté entre los ancestros de Z_i. Por otro lado, S_{MW} pone menos atención a la evidencia de lo que lo hace la distribución *a posteriori* $P(\mathbf{z}|\mathbf{e})$, ya que los valores muestreados para cada Z_i *ignoran* la evidencia que se encuentre entre los no-ancestros de Z_i[9].

El peso de verosimilitud w compensa la diferencia entre la distribución de muestreo real y la deseada. El peso para una muestra dada $\mathbf{x}$, formada por $\mathbf{z}$ y $\mathbf{e}$, es el producto de probabilidades de cada variable de evidencia dados sus padres (donde algunos o todos pueden estar entre los Z_i):

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | padres(E_i)) \tag{14.7}$$

[9] Idealmente, nos gustaría usar una distribución de muestreo igual a la auténtica *a posteriori* $P(\mathbf{z}|\mathbf{e})$, para tener en cuenta a toda la evidencia. Sin embargo, esto no puede hacerse eficientemente. Si se pudiera, podríamos aproximar la probabilidad deseada a una precisión arbitraria con un número polinomial de muestras. Se puede demostrar que tal esquema de aproximación en tiempo polinomial no puede existir.

Multiplicando las ecuaciones (14.6) y (14.7), vemos que la probabilidad *ponderada* de una muestra tiene una forma particularmente conveniente:

$$S_{MW}(\mathbf{z}, \mathbf{e})w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(y_i | padres(Y_i)) \prod_{i=1}^{m} P(e_i | padres(E_i)) = P(\mathbf{y}|\mathbf{e}), \qquad (14.8)$$

ya que los dos productos utilizan todas las variables de la red, lo que nos permite usar la Ecuación (14.1) de la probabilidad conjunta.

Ahora es fácil demostrar que las estimaciones obtenidas por la ponderación de la verosimilitud son consistentes. Para cualquier valor particular x de X, la probabilidad *a posteriori* estimada puede calcularse como sigue:

$$\begin{aligned}
\hat{P}(x|\mathbf{e}) &= \alpha \sum_{\mathbf{y}} N_{MW}(x, \mathbf{y}, \mathbf{e})w(x, \mathbf{y}, \mathbf{e}) && \text{por PONDERACIÓN-VEROSIMILITUD} \\
&\approx \alpha \sum_{\mathbf{y}} S_{MW}(x, \mathbf{y}, \mathbf{e})w(x, \mathbf{y}, \mathbf{e}) && \text{para un } N \text{ grande} \\
&\approx \alpha' \sum_{\mathbf{y}} P(x, \mathbf{y}, \mathbf{e}) && \text{por la Ecuación (14.8)} \\
&\approx \alpha' P(x, \mathbf{e}) = P(x|\mathbf{e})
\end{aligned}$$

Así, la ponderación de la verosimilitud retorna estimaciones consistentes.

Ya que la ponderación de la verosimilitud utiliza todas las muestras generadas, puede ser mucho más eficiente que el muestreo por rechazo. Adolecerá, sin embargo, una degradación en el rendimiento según aumenten el número de variables de evidencia. Puesto que la mayoría de las muestras tendrán pesos muy bajos, la estimación ponderada estará controlada más por el pequeñísimo porcentaje de muestras que concuerden con la evidencia que por la verosimilitud de dicha evidencia aunque ésta sea infinitesimal. El problema se agrava si las variables de evidencia aparecen las últimas en la ordenación de variables, porque entonces las muestras serán simulaciones que tienen poca similitud con la realidad sugerida por la evidencia.

Inferencia por simulación en cadenas de Markov

MONTE-CARLO PARA CADENAS DE MARKOV

En esta sección, explicaremos el algoritmo de **Monte Carlo para cadenas de Markov** (MCCM) para inferencia en redes Bayesianas. Primero describiremos lo que el algoritmo hace, después explicaremos por qué funciona y por qué tiene un nombre tan complicado.

El algoritmo MCCM

A diferencia de otros algoritmos de muestreo, que generan cada suceso a partir de cero, MCCM genera cada suceso haciendo un cambio aleatorio en el suceso precedente. Es por lo tanto útil pensar en la red como que está en un *estado actual* concreto mediante la especificación de un valor para cada variable. El siguiente estado se genera muestreando de forma aleatoria un valor de una de las variables de no-evidencia X_i, *condicionada a los valores actuales de las variables en el manto de Markov de* X_i. (Recuerde del Apartado 14.2 que el manto de Markov para una variable consta de sus padres, sus hijos, y los padres de los hijos). El MCCM por tanto deambula de forma aleatoria alrededor del espacio de estados (el espacio de las posibles asignaciones com-

pletas) cambiando una variable en cada momento, pero manteniendo fijas la variables de evidencia.

Considere la pregunta **P**(*Llueve*|*Aspersor = verdad, HierbaHúmeda = verdad*) aplicada a la red de la figura 14.11(a). Las variables de evidencia *Aspersor* y *HierbaHúmeda* se fijan a sus valores observados y las variables ocultas *Nublado* y *Lluvia* se inicializan de forma aleatoria (supongamos *verdad* y *falso*, respectivamente). Así, el estado inicial es [*verdad, verdad, falso, verdad*]. Ahora se llevan a cabo los siguientes pasos repetidamente:

1. Se muestrea *Nublado*, dados los valores actuales de las variables de su manto de Markov: en este caso muestreamos a partir de **P**(*Nublado*|*Aspersor = verdad, Lluvia = falso*). (Dentro de poco, expondremos cómo calcular esta distribución). Supongamos que el resultado es *Nublado = falso*. Entonces el nuevo estado actual es [*falso, verdad, falso, verdad*].
2. Se muestrea *Lluvia*, dados los valores actuales de las variables de su manto de Markov: en este caso muestreamos a partir de **P**(*Lluvia*|*Nublado = falso, Aspersor = verdad, HierbaHúmeda = verdad*). Suponga que esto da *Lluvia = verdad*. El nuevo estado actual es [*falso, verdad, verdad, verdad*].

Cada estado visitado durante este proceso es una muestra que contribuye a la estimación de la variable pregunta *Lluvia*. Si el proceso visita 20 estados en los que *Lluvia* es verdad y 60 estados donde *Lluvia* es falso, entonces la respuesta a la cuestión es NORMALIZAR($\langle 20, 60\rangle$) = $\langle 0{,}25, 0{,}75\rangle$. El algoritmo completo se muestra en la Figura 14.15.

```
función PREGUNTAR-MCCM (X, e, rb, N) devuelve una estimación de P(X|e)
  entradas: N[X], un vector de contadores sobre X, inicialmente nulos
            Z, las variables no-evidencia de la rb
            x, el estado actual de la red, inicialmente copiado de e
            rb, una red bayesiana
            N, el número total de muestras a generar

  inicializar x con valores aleatorios para las variables de Z
  desde j = 1 hasta N hacer
    N[x] ← N[x] + 1 donde x es el valor de X en x
    para cada Z_i de Z hacer
      muestrear el valor de Z_i en x a partir de P(Z_i|manto(Z_i)) dados los valores de Manto(Z_i) en x
  devolver NORMALIZAR(N[X])
```

Figura 14.15 El algoritmo MCCM para inferencia aproximada en redes bayesianas.

Por qué funciona MCCM

Mostraremos ahora que el MCCM devuelve estimaciones consistentes de las probabilidades posteriores. El contenido de esta sección es bastante técnico, pero la argumentación básica es sencilla: *el proceso de muestreo tiende a un equilibrio dinámico en el cual el porcentaje de ejecución del tiempo empleado en cada estado es exactamente pro-*

PROBABILIDAD DE TRANSICIÓN

porcional a su probabilidad a posteriori. Esta sorprendente propiedad se deduce a partir de la **probabilidad de transición** concreta con la que el proceso se mueve de un estado a otro, la cual está definida como la distribución condicionada de la variable que se esté muestreando dado su manto de Markov.

CADENA DE MARKOV

Sea $q(\mathbf{x} \rightarrow \mathbf{x}')$ la probabilidad de que el proceso haga una transición del estado $\mathbf{x}$ al estado $\mathbf{x}'$. Esta probabilidad de transición define lo que se llama una **cadena de Markov** del espacio de estados (las cadenas de Markov serán tratadas en profundidad en los capítulos 15 y 17). Ahora supongamos que realizamos una cadena de Markov de t pasos, y sea $\pi_t(\mathbf{x})$ la probabilidad de que el sistema esté en el estado $\mathbf{x}$ en el instante t. Análogamente, sea $\pi_{t+1}(\mathbf{x}')$ la probabilidad de estar en el estado $\mathbf{x}'$ en el momento $t + 1$. Dada $\pi_t(\mathbf{x})$, podemos calcular $\pi_{t+1}(\mathbf{x}')$ como una sumatoria, en todos los estados en los que el sistema podría estar en el instante t, de la probabilidad de estar en ese estado multiplicada por la probabilidad de hacer una transición al estado $\mathbf{x}'$:

$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}')$$

DISTRIBUCIÓN ESTACIONARIA

Diremos que la cadena ha alcanzado su **distribución estacionaria** si $\pi_t = \pi_{t+1}$. Llamemos π a esta distribución estacionaria; su ecuación que la define es así

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \qquad \text{para todo } \mathbf{x}' \tag{14.9}$$

Bajo ciertos supuestos estándares referentes a la distribución de probabilidad de transición q[10], hay exactamente una distribución π que satisface esta ecuación para cualquier q dada.

La ecuación (14.9) se puede leerse diciendo que si un estado se contempla como la situación actual de una «población», entonces el «abandono» esperado de cada estado es igual a la «afluencia» esperada de todos los estados. Una manera lógica para satisfacer esta relación es que el flujo esperado entre cualquier par de estados sea el mismo en ambas direcciones. Esta es la propiedad de **balance detallado**:

BALANCE DETALLADO

$$\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) \qquad \text{para todo } \mathbf{x}, \mathbf{x}' \tag{14.10}$$

Podemos demostrar que el balance detallado implica estacionamiento simplemente sumando en $\mathbf{x}$ la ecuación (14.10). Tenemos que

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}')$$

donde el último paso se deduce porque está garantizado que ocurra una transición desde $\mathbf{x}'$.

Ahora mostraremos que la probabilidad de transición $q(\mathbf{x} \rightarrow \mathbf{x}')$ definida en el paso de muestreo de PREGUNTAR-CMMC satisface la ecuación del balance detallado con una distribución estacionaria igual a $P(\mathbf{x}|\mathbf{e})$, (la auténtica distribución *a posteriori* de las variables ocultas). Haremos esto en dos pasos. Primero, definiremos una cadena de Markov en la que cada variable es muestreada pero condicionada a los valores actuales de *todas* las demás variables, y demostraremos que esto satisface el balance detallado. En-

[10] La cadena de Markov definida por q debe ser **ergódica**; esto es, en esencia, que cada estado debe ser alcanzable desde cualquier otro, y no puede haber ciclos estrictamente periódicos.

tonces, sencillamente observaremos que, para redes Bayesianas, hacer eso es igual a muestrear pero condicionando a las variables del manto de Markov (*véase* Apartado 14.2).

Sea X_i la variable a muestrear, y sean $\overline{\mathbf{X}}_i$ todas las variables ocultas *distintas de* X_i. Sus valores en el estado actual son x_i y $\overline{\mathbf{x}}_i$. Si muestreamos un nuevo valor x'_i para X_i pero condicionando a todas las demás variables, incluida la evidencia, tenemos

$$q(\mathbf{x} \rightarrow \mathbf{x}') = q((x_i, \overline{\mathbf{x}}_i) \rightarrow (x'_i, \overline{\mathbf{x}}_i) = P(x'_i | \overline{\mathbf{x}}_i, \mathbf{e})$$

MUESTREADOR DE GIBBS

Esta probabilidad de transición se llama **muestreador**[11] **de Gibbs** y es una expresión particularmente conveniente del MCCM. Ahora demostraremos que el muestreador de Gibbs verifica la condición de balance detallado con la auténtica distribución *a posteriori*:

$$\begin{aligned}
\pi(\mathbf{x})q(\mathbf{x} \rightarrow \mathbf{x}') &= P(\mathbf{x}|\mathbf{e})P(x_i|\overline{\mathbf{x}}_i, \mathbf{e}) = P(x'_i, \overline{\mathbf{x}}_i|\mathbf{e})P(x'_i|\overline{\mathbf{x}}_i, \mathbf{e}) \\
&= P(x_i|\overline{\mathbf{x}}_i, \mathbf{e})P(\overline{\mathbf{x}}_i|\mathbf{e})P(x'_i|\overline{\mathbf{x}}_i, \mathbf{e}) \quad \text{(usando la regla de la cadena en el primer término)} \\
&= P(x_i|\overline{\mathbf{x}}_i, \mathbf{e})P(x'_i, \overline{\mathbf{x}}_i|\mathbf{e}) \quad \text{(usando la regla de la cadena hacia atrás)} \\
&= \pi(\mathbf{x}')q(\mathbf{x}' \rightarrow \mathbf{x})
\end{aligned}$$

Como se dijo en el Apartado 14.2, una variable es independiente de todas las demás variables dado su manto de Markov; así,

$$P(x'_i|\overline{\mathbf{x}}_i, \mathbf{e}) = P(x'_i|manto(X_i))$$

donde $manto(X_i)$ denota a los valores de las variables del manto de Markov de X_i, $Manto(X_i)$. Como se indica en el Ejercicio 14.10, la probabilidad de una variable dado su manto de Markov es proporcional a la probabilidad de la variable dados sus padres multiplicada por la probabilidad de cada hijo dados sus padres respectivos:

$$P(x'_i|manto(X_i)) = \alpha\, P(x'_i|padres(X_i)) \times \prod_{Y_j \in Hijos(X_i)} P(y_j|padres(Y_j)) \qquad (14.11)$$

Así, para cambiar cada variable X_i, el número de multiplicaciones que se necesitan es igual al número de hijos de X_i.

Hemos discutido aquí una variante sencilla del MCCM, a saber, el muestreador de Gibbs. En su forma más general, el MCCM es un método potente para la computación usando modelos probabilistas y se han desarrollado muchas variantes, incluido el algoritmo de enfriamiento simulado que se presentó en el Capítulo 4, los algoritmos de satisfactibilidad estocástica del Capítulo 7, y el muestreador de Metrópolis-Hastings del Capítulo 15.

14.6 Extensión de la probabilidad a representaciones de primer orden

En el Capítulo 8, explicamos las ventajas para la representación de la lógica de primer orden en comparación con la lógica proposicional. La lógica de primer orden se desti-

[11] El término *muestreador* no existe en castellano y es una traducción demasiado literal del termino inglés *sampler*. En realidad, debería decirse algo así como *procedimiento para obtener muestras*, pero este tipo de expresiones no suelen encontrase en la literatura. *(Nota del traductor.)*

na a la existencia de objetos y a las relaciones entre ellos y puede expresar hechos referidos a *algunos* o a *todos* los objetos de un dominio. Esto a menudo se traduce en representaciones que son enormemente más concisas que las descripciones equivalentes usando proposiciones. Ahora, las redes Bayesianas son en esencia proposicionales: el conjunto de variables es fijo y finito, y cada una tiene un domino fijo de posibles valores. Este hecho limita el campo de aplicación de las redes Bayesianas. *Si podemos encontrar un modo de combinar la teoría de la probabilidad con la potencia expresiva de las representaciones de primer orden, cabe esperar que seamos capaces de incrementar espectacularmente el rango de problemas que se puedan manejar*.

La visión básica necesaria para alcanzar este objetivo es la siguiente: en el contexto proposicional, una red Bayesiana determina probabilidades sobre sucesos atómicos, cada uno de los cuales especifica un valor para cada variable de la red. Así, un suceso atómico es un **modelo** o **mundo posible**, en la terminología de la lógica proposicional. En el contexto de primer orden, un modelo (con su interpretación) especifica un dominio de objetos, las relaciones que se verifican entre esos objetos, y una correspondencia desde las constantes y predicados de la base de conocimiento hasta los objetos y relaciones en el modelo. Por lo tanto, *una base de conocimiento probabilista de primer orden debería especificar probabilidades para todos los posibles modelos de primer orden*. Sea $\mu(M)$ la probabilidad asignada al modelo M por la base de conocimiento. Para cualquier oración de primer orden ϕ, la probabilidad $P(\phi)$ viene dada de la forma usual sumando en todos los mundos posibles en los que ϕ es cierta:

$$P(\phi) = \sum_{M\,:\,\phi \text{ es cierta en } M} \mu(M) \tag{14.12}$$

Hasta el momento, todo va bien. Sin embargo, hay un problema: el conjunto de modelos de primer orden es infinito. Esto significa que (1) calcular la suma total podría ser impracticable, y (2) la especificación de una distribución completa y consistente definida sobre un conjunto infinito de mundos podría ser muy difícil.

Recortemos nuestras aspiraciones, por lo menos por ahora. En particular, ideemos un lenguaje restringido para el que haya sólo muchos (finitos) modelos de interés. Hay varias formas de hacer esto. Aquí, presentamos los **modelos probabilistas relacionales**, o MPR, los cuales toman prestadas algunas ideas de las redes semánticas (Capítulo 10) y de las bases de datos relacionales orientadas a objetos. Se discuten otras aproximaciones en las notas bibliográficas e históricas.

MODELO PROBABILISTA RELACIONAL

Los MPR dan cabida a los símbolos constantes para los objetos relativos a nombres. Por ejemplo, sea *ProfSmith* el nombre de un profesor, y sea *Jones* el nombre de un estudiante. Cada objeto es una instancia de una clase; por ejemplo *ProfSmith* es un *Profesor* y *Jones* es un *Estudiante*. Asumimos que la clase de cada símbolo constante es conocida.

FUNCIÓN SIMPLE

Nuestros símbolos de funciones se dividirán en dos tipos. El primero de ellos, **funciones simples**, que asocian un objeto a un valor de un dominio fijo de valores, y no a otro objeto estructurado, exactamente como una variable aleatoria. Por ejemplo *Inteligencia*(*Jones*) y *Financiación*(*ProfSmith*) podría ser *alto* o *bajo*; *Éxito*(*Jones*) y *Prestigio*(P*rofSmith*) puede ser *verdad* o *falso*. Los símbolos de funciones no deben aplicarse a valores como *verdad* y *falso*, así no es posible tener anidamientos de funciones simples. De este modo, evitamos una fuente de infinidades. El valor de una función aplica-

da a un objeto dado puede ser observado o desconocido; estas serán las variables aleatorias básicas de nuestra representación[12].

FUNCIÓN COMPLEJA

También permitimos **funciones complejas** que asocian objetos a otros objetos. Por ejemplo *Consultor*(*Jones*) podría ser *ProfSmith*. Cada función compleja tiene un dominio específico y un rango, los cuales son clases. Por ejemplo, el dominio de *Consultor* es *Estudiante* y el rango es *Profesor*. Las funciones sólo se aplican a la clase correcta; por ejemplo, el *Consultor* de *ProfSmith* no está definido. Las funciones complejas pueden estar anidadas: *DirectorDpto*(*Consultor*(*Jones*)) podría ser *ProfMoore*. Supondremos (por ahora) que los valores de todas las funciones complejas son conocidos para todos los símbolos constantes. Ya que la base de conocimiento es finita, esto implica que cada concatenación de llamadas a funciones complejas conduce a uno de los objetos, que están en un número finito[13].

El último elemento que necesitamos es la información probabilista. Para cada función simple, especificamos un conjunto de padres, al igual que en redes Bayesianas. Los padres pueden ser otras funciones simples del mismo objeto; por ejemplo, la *Financiación* de un *Profesor* podrían considerar su *Éxito*. Los padres también pueden ser funciones simples de objetos *relacionados* (por ejemplo, el *Éxito* de un *Estudiante* podría depender de la *Inteligencia* del estudiante y del *Prestigio* del consultor del estudiante). Éstas son realmente afirmaciones *cuantificadas universalmente* referentes a los padres de todos los objetos de una clase. Así, podríamos escribir

$$\forall x \;\; x \in Estudiante \;\Rightarrow \\ Padres(Éxito(x)) = \{Inteligencia(x), Prestigio(Consultor(x))\}$$

(Aunque menos formal, podemos dibujar diagramas como el de la Figura 14.16(a).) Ahora establecemos la distribución de probabilidad condicionada para los hijos, dados sus padres. Por ejemplo, podríamos decir que

$$\forall x \;\; x \in Estudiante \;\Rightarrow \\ P(Éxito(x) = verdad \mid Inteligencia(x) = alto, Prestigio(Consultor(x)) = verdad) = 0{,}95$$

Como en redes semánticas, podemos añadir la distribución condicionada a la clase en sí, para que las instancias **hereden** las dependencias y probabilidades condicionadas de la clase.

La semántica del lenguaje de los MPR supone que cada símbolo constante se refiere a un objeto distinto, **asunción de nombres únicos** descrito en el Capítulo 10. Dada esta suposición y las restricciones enumeradas anteriormente, se puede demostrar que cada MPR genera un conjunto finito y fijo de variables aleatorias, cada una de las cuales es una función simple aplicada a un símbolo constante. Entonces, supuesto que las dependencias padre-hijo son *acíclicas*, podemos construir una red Bayesiana equivalente. Esto es, el MPR y la red Bayesiana especifican probabilidades idénticas para cada mundo posible. La Figura 14.16(b) muestra la red Bayesiana correspondiente al MPR

[12] Representan un papel muy similar al de las sentencias atómicas generadas en el proceso de proposicionalización descrito en la Sección 9.1.

[13] Esta restricción significa que no podemos usar funciones complejas tales como *Padre* o *Madre*, que conducen a encadenamientos potencialmente infinitos que tendrían al final un objeto desconocido. Revisamos luego esta restricción.

Figura 14.16 (a) Un MPR describiendo dos clases: *Profesor* y *Estudiante*. Hay dos profesores y dos estudiantes, y *ProfSmith* es un consultor de ambos estudiantes. (b) La red bayesiana equivalente al MPR de (a).

de la Figura 14.16(a). Observe que los enlaces del *Consultor* en el MPR no están en la red Bayesiana. Esto es porque son fijos y conocidos. Aparecen *implícitamente* en la red topológica, sin embargo, por ejemplo, *Éxito*(*Jones*) tiene a *Prestigio*(*ProfSmith*) como padre ya que el *Consultor*(*Jones*) es *ProfSmith*. En general, las relaciones que se cumplen entre los objetos determinan el patrón de dependencias entre las propiedades de esos objetos.

DEPENDENCIA RECURSIVA

Hay varios modos de incrementar la potencia expresiva de los MPR. Podemos permitir **dependencias recursivas** entre las variables para captar determinados tipos de relaciones recurrentes. Por ejemplo, suponga que la adicción a la comida rápida está causada por el *GenMc*. Entonces, para cualquier x, *GenMc*(x) depende de *GenMc*(*Padre*(x)) y de *GenMc*(*Madre*(x)), que dependen a su vez de *GenMc*(*Padre*(*Padre*(x))), *GenMc*(*Madre*(*Padre*(x))) y así sucesivamente. Incluso aunque tales bases de conocimientos correspondan a redes Bayesianas con muchas (infinitamente) variables aleatorias, las soluciones pueden obtenerse, a veces, a partir de ecuaciones de punto fijo. Por ejemplo, la distribución de equilibrio del *GenMc* puede calcularse, dada la probabilidad condicionada de la herencia. Otra familia muy importante de bases de conocimiento recursivas consiste en los **modelos de probabilidad temporales** descritos en el Capítulo 15. En estos modelos, las propiedades de un estado en el momento t dependen de las propiedades del estado en el momento $t - 1$ y así sucesivamente.

INCERTIDUNBRE RELACIONAL

Los MPR también pueden extenderse para permitir **incertidumbre relacional**, esto es, incertidumbre referente a los valores de funciones complejas. Por ejemplo, podemos no saber quién es *Consultor*(*Jones*). *Consultor*(*Jones*) se convierte entonces en una variable aleatoria, con posibles valores *ProfSmith* y *ProfMoore*. La correspondiente red se muestra en la Figura 14.17.

INCERTUDUMBRE DE IDENTIDAD

Puede haber también **incertidumbre de identidad**; por ejemplo, podríamos no saber si *Mary* y *ProfSmith* son la misma persona. Con incertidumbre de identidad, el número de objetos y proposiciones puede variar a lo largo de los mundos posibles. Un mundo donde *Mary* y *ProfSmith* son la misma persona tiene un objeto menos que un mundo en el que son personas diferentes. Esto hace el proceso de inferencia más complicado,

Figura 14.17 Parte de una red bayesiana correspondiente a un MPR en el que *Consultor(Jones)* es desconocido, pero o es *ProfSmith* o es *ProfMoore*. La elección del consultor depende de cuánta financiación tenga cada profesor. Nótese que *Éxito(Jones)* ahora dependerá del *Prestigio* de *ambos* profesores, aunque el valor de *Consultor(Jones)* determina cuál tiene influencia en realidad.

pero el principio básico establecido en la Ecuación (14.12) todavía se verifica: la probabilidad de cualquier oración está bien definida y puede ser calculada. La incertidumbre de identidad es especialmente importante para robots y para sistemas sensoriales embebidos que deben estar al tanto de múltiples objetos. Volveremos a este problema en el Capítulo 15.

Ahora examinemos el asunto de la inferencia. Claramente, la inferencia se puede realizar en la red Bayesiana equivalente, supuesto que restringimos el lenguaje del MPR para que así la red equivalente sea finita y tenga una estructura fija. Esto es análogo al modo en que puede hacerse la inferencia lógica de primer orden, vía inferencia proposicional sobre la base de conocimiento proposicional equivalente. (*Véase* Apartado 9.1.) Como en el caso lógico, la red equivalente podría ser demasiado grande para construirla, sólo se evalúa. Las interconexiones densas son también un problema. (*Véase* Ejercicio 14.12.) Algoritmos aproximados como el MCCM (Sección 14.5) son así muy útiles para la inferencia en MPR.

Cuando se aplica el MCCM a la red Bayesiana equivalente para una sencilla base de conocimiento de un MPR sin incertidumbre relacional o de identidad, el algoritmo muestrea en el espacio de posibles mundos que están definidos por los valores de funciones simples de los objetos. Es fácil ver que esta aproximación puede extenderse para manejar adicionalmente incertidumbre relacional y de identidad. En ese caso, una transición entre mundos posibles podría cambiar el valor de una función simple o podría cambiar una función compleja, y así conducir a una transformación en la estructura de dependencias. Las transiciones podrían también cambiar las relaciones de identidad entre los símbolos constantes. Así, el MCCM parece ser un modo elegante para tratar la inferencia en bases de conocimiento probabilistas de primer orden que se sean realmente expresivas.

La investigación en esta área está todavía en un estado primitivo, pero está mostrando claramente que el razonamiento probabilista de primer orden produce un incremento tremendo en la efectividad de los sistemas de IA en el tratamiento de la información incierta. Aplicaciones potenciales incluyen visión por computador, comprensión del lenguaje natural, recuperación de información y valoración de situaciones. En todas es-

tas áreas, el conjunto de objetos (y así el conjunto de variables aleatorias) no es conocido con antelación, por lo que los métodos puramente «proposicionales», tales como redes Bayesianas, son incapaces de representar la situación en su totalidad. Estas áreas se han expandido al buscar en el espacio de modelos, pero los MPRs permiten el razonamiento sobre esta incertidumbre con un modelo sencillo.

14.7 Otros enfoques al razonamiento con incertidumbre

Otras ciencias (por ejemplo, la Física, la Genética, y la Economía) han favorecido bastante a la probabilidad como un modelo para la incertidumbre. En 1819, Pierre Laplace dijo «la teoría de la probabilidad no es nada, solamente el sentido común para el cálculo». En 1850, James Maxwell dijo «la lógica auténtica para este mundo es el cálculo de probabilidades, que tiene en cuenta la grandeza de la probabilidad que está, o debería estar, en la mente de un hombre sensato».

Dada esta larga tradición, es quizá sorprendente que la IA haya considerado muchas alternativas a la probabilidad. Los primeros sistemas expertos de la década de 1970 ignoraron la incertidumbre y usaron razonamiento estrictamente lógico, pero pronto llegó a estar claro que esto era poco práctico para la mayoría de los dominios del mundo real. La siguiente generación de sistemas expertos (especialmente en dominios médicos) usaron técnicas probabilistas. Los resultados iniciales fueron prometedores, pero no podían generalizarse debido al número exponencial de probabilidades necesarias en la distribución conjunta de probabilidad. (Por entonces se desconocían los algoritmos eficientes en redes bayesianas.) Como una consecuencia, el enfoque probabilista descendió su aceptación aproximadamente entre 1975 y 1988, y se intentaron diversas alternativas a la probabilidad por distintas razones:

- Un punto de vista común es que la teoría de la probabilidad es esencialmente numérica, mientras que el razonamiento humano es más «cualitativo». Ciertamente, no nos damos cuenta conscientemente en realizar cálculos numéricos de grados de creencia. (Ni somos conscientes de hacer unificaciones, a pesar de eso parece que somos capaces de algún tipo de razonamiento lógico.) Debe ser que tenemos algún tipo de grados de creencia numéricos codificados directamente en la fortaleza de las conexiones y activaciones de nuestras neuronas. (En ese caso, la dificultad para el acceso consciente a esas fortalezas no es sorprendente.) Debería también notar que los mecanismos de razonamiento cualitativo pueden construirse directamente por encima de la teoría de la probabilidad, por lo que el argumento de «no números» contra la probabilidad tiene poca fuerza. No obstante, algunos esquemas cualitativos tienen un buen montón de atractivos por propio derecho. Uno de los mejores estudiados es el **razonamiento por defecto**, que trata las conclusiones no como «creídas hasta un cierto grado», sino como «creídas hasta que se encuentre un razón mejor para creer alguna otra cosa». El razonamiento por defecto se trató en el Capítulo 10.

- Los enfoques **basados en reglas** para la incertidumbre también se han probado. Tales aproximaciones esperan agregar los sucesos de los sistemas lógicos basados en reglas, pero añaden un tipo de «factor amañado» a cada regla para reubicar la incertidumbre. Estos métodos se desarrollaron a mediados de 1970 y constituyeron las bases de un gran número de sistemas expertos en medicina y otras áreas.
- Una área que no habíamos tratado hasta este momento es la cuestión de la **ignorancia**, como opuesto a la incertidumbre. Considere el lanzamiento de una moneda. Si sabemos que la moneda no está trucada, entonces una probabilidad de 0,5 para la cara es razonable. Si sabemos que la moneda está trucada, pero no sabemos qué lado, entonces 0,5 es la única probabilidad razonable. La **teoría de Dempster-Shafer** usa grados de creencia dados por **intervalos de valores** para representar el conocimiento de un agente referente a la probabilidad de una proposición. También se plantean otros métodos utilizando probabilidades de segundo orden.
- Las probabilidades hacen la misma misión ontológica que la lógica: los sucesos son ciertos o falsos en el mundo, incluso si el agente está dudoso para determinar cuál es el caso. Los investigadores en **lógica borrosa** han propuesto una ontología que permite la **vaguedad**: que un suceso pueda ser «en cierto modo» verdad. La vaguedad y la incertidumbre son de hecho asuntos ortogonales, como veremos.

Los siguientes tres apartados tratan algunas de estas aproximaciones con un poco más de profundidad. No daremos material técnico detallado, pero citamos referencias para más información.

Métodos basados en reglas para razonamiento con incertidumbre

Los sistemas basados en reglas surgieron con los primeros trabajos en sistemas prácticos e intuitivos con inferencia lógica. Los sistemas lógicos en general, y los sistemas basados en reglas lógicas en particular, tienen tres propiedades deseables:

LOCALIDAD

— **Localidad:** en los sistemas lógicos, siempre que tenemos una regla de la forma $A \Rightarrow B$, podemos concluir B, dada la evidencia A, *sin preocuparnos de lo que le ocurra a cualquiera de las otras reglas*. En sistemas probabilistas, necesitamos considerar todas las evidencias del manto de Markov.

DESAPEGO

— **Desapego:** una vez que se encuentra una demostración lógica para una proposición B, la proposición puede ser utilizada independientemente de cómo se deduzca. Esto es, puede *desapegarse* de su justificación. En relación con las probabilidades, por otro lado, el origen de la evidencia para una creencia es importante para el razonamiento subsiguiente.

OPERATIVIDAD DE LA VERDAD

— **Operatividad de la verdad:** en lógica, la verdad de las oraciones complejas puede calcularse a partir de la verdad de sus componentes. La combinación probabilista no trabaja de este modo, excepto bajo condiciones fuertes de independencia global.

Ha habido distintos intentos para diseñar esquemas de razonamiento con incertidumbre que mantengan estas ventajas. La idea es ligar grados de creencia a proposiciones y re-

glas e idear esquemas puramente lógicos para la combinación y propagación de estos grados de creencia. Los esquemas son también operativos de la verdad; por ejemplo, el grado de creencia de $A \vee B$ es una función de la creencia de A y la creencia de B.

Las malas noticias para los sistemas basados en reglas es que las propiedades de *localidad, desapego y operativos de la verdad simplemente no son congruentes con el razonamiento incierto*. Veamos primero la operatividad de la verdad. Sea H_1 el suceso de que al lanzar una moneda no trucada aparezca cara, sea T_1 el suceso de que aparezca cruz en algún lanzamiento, y sea H_2 el suceso de que aparezca cara en el segundo lanzamiento. Claramente, los tres sucesos tienen la misma probabilidad, 0,5, y así un sistema que use operatividad de la verdad debe asignar la misma creencia a la disyunción de cualquiera dos de ellos. Pero podemos ver que la probabilidad de la disyunción depende de los sucesos en sí y no de sus probabilidades:

$P(A)$	$P(B)$	$P(A \vee B)$
$P(H_1) = 0{,}5$	$P(H_1) = 0{,}5$ $P(T_1) = 0{,}5$ $P(H_2) = 0{,}5$	$P(H_1 \vee H_1) = 0{,}50$ $P(H_1 \vee T_1) = 1{,}00$ $P(H_1 \vee H_2) = 0{,}75$

Peor es cuando encadenamos la evidencia. Los sistemas con operatividad de la verdad tienen **reglas** de la forma $A \mapsto B$ que nos permite computar la creencia en B como una función de la creencia en la regla y la creencia en A. Se pueden diseñar tanto para los sistemas de encadenamiento hacia delante como en los de encadenamiento hacia atrás. La creencia en la regla se supone que es constante y se determina usualmente por el ingeniero de conocimiento, por ejemplo, como $A \mapsto_{0,9} B$.

Considere la situación de la hierba húmeda de la Figura 14.11(a). Si queremos ser capaces de hacer tanto razonamiento causal como de diagnóstico, necesitaríamos las dos reglas

$$Lluvia \mapsto HierbaHúmeda \qquad \text{y} \qquad HierbaHúmeda \mapsto Lluvia$$

Estas dos reglas forman un ciclo retroalimentado: la evidencia para *Lluvia* incrementa la creencia en *HierbaHúmeda*, lo que a su vez incrementa la creencia en *Lluvia* incluso más. Claramente, los sistemas de razonamiento con incertidumbre necesitan estar al tanto de los caminos a través de los que se propagó la evidencia.

El razonamiento intercausal (o justificación mediante razones convincentes) es también difícil. Reflexione lo que ocurre cuando tenemos las dos reglas

$$Aspersor \mapsto HierbaHúmeda \qquad \text{y} \qquad HierbaHúmeda \mapsto Lluvia$$

Suponga que vemos que el aspersor está encendido. Encadenando hacia delante por nuestras reglas, esto incrementa la creencia de que la hierba está húmeda, lo que a su vez incrementa la creencia de que está lloviendo. Pero esto es ridículo: el hecho de que el aspersor esté encendido justifica el que la hierba esté húmeda y debería *reducir* la creencia en la lluvia. Un sistema con operatividad de la verdad actúa como si también considerara la regla $Aspersor \mapsto Lluvia$.

Dadas estas dificultades, ¿cómo es posible que los sistemas con operatividad de la verdad se considerasen incluso útiles? La respuesta se encuentra en la restricción de la

FACTORES DE CERTEZA

tarea y en una ingeniería escrupulosa en la base de reglas de modo que las interacciones no deseables no ocurran. El ejemplo más famoso de sistemas con operatividad de la verdad para razonamiento incierto es el modelo de **factores de certeza**, que se desarrolló en el programa de diagnóstico médico MYCIN y fue ampliamente utilizado en sistemas expertos de finales de los 70 y 80. Casi todos los usos de factores de certeza involucraron conjuntos de reglas o eran puramente de diagnóstico (como en MYCIN) o eran puramente causales. Más aún, la evidencia sólo se introducía en las «raíces» del conjunto de reglas, y la mayoría de las reglas eran de conexión simple. Heckerman (1986) ha demostrado que, bajo estas circunstancias, una variación de poca importancia en la inferencia con factores de certeza era exactamente equivalente a la inferencia bayesiana en poliárboles. En otras circunstancias, los factores de certeza podrían producir catastróficamente grados incorrectos de creencia por un sobrerecuento de la evidencia. Cuando los conjuntos de reglas se hacen voluminosos, llegan a ser más comunes las interacciones indeseables entre las reglas, y los profesionales ven que los factores de certeza de muchas otras reglas tienen que ser «pellizcadas» cuando se añaden nuevas reglas. Es innecesario decir que este enfoque ya no se recomienda.

Representación de la ignorancia: teoría de Dempster-Shafer

DEMPSTER-SHAFER

FUNCIÓN DE CREENCIA

La teoría de **Dempster-Shafer** está diseñada para tratar con la diferencia entre **incertidumbre** e **ignorancia**. Más que calcular la probabilidad de una proposición, calcula la probabilidad con que la evidencia respalda la proposición. Esta medida de creencia se llama **función de creencia**, y se escribe $Bel(X)$.

Como ejemplo de función de creencia, volvamos al lanzamiento de monedas. Suponga que un personaje sombrío de pronto va hacia usted y le ofrece apostar 10 dólares a que en su moneda aparecerá cara en el siguiente lanzamiento. Dado que la moneda puede o no puede estar trucada, ¿qué creencia debería asignar al suceso de que aparezca cara? La teoría de Dempster-Shafer dice que puesto que no tiene ninguna evidencia de ningún modo, tiene que decir que la creencia $Bel(Cara) = 0$ y también que $Bel(\neg Cara) = 0$. Esto hace escépticos a los sistemas de razonamiento de Dempster-Shafer, de un modo que tiene cierto atractivo intuitivo. Ahora suponga que tiene un experto a su disposición que atestigua con una certidumbre del 90 por ciento que la moneda es legal (es decir, está seguro en un 90 por ciento de que $P(Cara) = 0{,}5$). Entonces, la teoría de Dempster-Shafer asigna $Bel(Cara) = 0{,}9 \times 0{,}5 = 0{,}45$ y de igual modo $Bel(\neg Cara) = 0{,}45$. Hay todavía un porcentaje de 10 que indica una «laguna» que no está explicada por la evidencia. La «regla de Dempster» (Dempster, 1968) muestra cómo combinar la evidencia para asignar un valor nuevo para Bel, y el trabajo de Shafer extiende esto a un modelo computacional completo.

Como con razonamiento por defecto, hay un problema para conectar las creencias a las acciones. Con probabilidades, la teoría de la decisión dice que si $P(Cara) = P(\neg Cara) = 0{,}5$, entonces (suponiendo que el ganar 10 dólares y perder 10 dólares se consideran opuestas de igual magnitud) quien lo razone será indiferente entre la acción de la aceptación y de la declinación de la apuesta. Quien razone según Dempster-Shafer

tiene que $Bel(\neg Cara) = 0$ y así no hay razón para aceptar la apuesta, pero además también tiene $Bel(Cara) = 0$ y así no hay razón para declinarla. Así, parece que quien razone según Dempster-Shafer llega a la misma conclusión sobre cómo actuar en este caso. Desafortundamente, la teoría de Dempster-Shafer no permite decisiones claras en muchos otros casos donde la inferencia probabilista elabora una elección precisa. De hecho, la noción de utilidad en el modelo de Dempster-Shafer no se comprende todavía bien.

Una interpretación de la teoría de Dempster-Shafer es que define un intervalo de probabilidad: el intervalo para *Cara* es [0, 1] antes de nuestro testimonio experto y [0,45, 0,55] después. El ancho del intervalo debería ser una ayuda en la decisión de cuándo necesitamos adquirir más evidencia: puede decirle que el testimonio del experto le ayudará si no sabe si la moneda es legal, pero no le ayudará si se ha enterado de que la moneda es legal. Sin embargo, no hay una directriz clara de cómo hacer esto, porque no está claro el significado de qué quiere decir la anchura del intervalo. En el enfoque bayesiano, este tipo de razonamiento puede hacerse fácilmente estudiando cuánta creencia debería cambiar si estaba dispuesto a adquirir más evidencia. Por ejemplo, el saber que la moneda es legal tendría un impacto significativo sobre la creencia de que saldrá cara, y la detección de un peso asimétrico tendría un impacto sobre la creencia de que la moneda es legal. Un modelo completo bayesiano incluiría estimaciones probabilistas de factores como éstos, permitiéndonos expresar nuestra «ignorancia» en términos de cómo nuestras creencias cambiarían ante la recogida de información futura.

Representación de la vaguedad: conjuntos difusos y lógica difusa

TEORÍA DE LOS CONJUNTOS DIFUSOS

La **teoría de los conjuntos difusos** es un instrumento para la especificación de lo bien que un objeto satisface una descripción vaga. Por ejemplo, considere la proposición «Nate es alta». ¿Es esto verdad si Nate mide 5′ 10″? La mayoría de la gente vacilaría al responder «verdad» o «falso», preferirían decir, «tipo de». Nótese que esto no es una pregunta de incertidumbre sobre el mundo externo (estamos seguros de la altura de Nate). El asunto es que el término lingüístico «alto» no se refiere a una delimitación clara de los objetos en dos clases (hay *grados* de altura). Por esta razón, *la teoría de los conjuntos difusos no es un método para razonamiento incierto en absoluto*. Mejor dicho, la teoría de los conjuntos difusos trata *Alto* como un predicado difuso y dice que el valor de verdad de *Alto*(*Nate*) es un número entre 0 y 1, en vez de un *verdad* o *falso* exacto. El nombre «conjunto difuso» proviene de la interpretación del predicado como la definición implícita de un conjunto de sus miembros (un conjunto que no tiene fronteras delimitadas).

LÓGICA DIFUSA

La **lógica difusa** es un método para el razonamiento con expresiones lógicas que describen las pertenencias a los conjuntos difusos. Por ejemplo, la oración compleja $Alto(Nate) \wedge Grueso(Nate)$ tiene un valor de verdad difuso que es una función de los valores de verdad de sus componentes. Las reglas estándares para la evaluación de la verdad borrosa, T, de una oración compleja son:

$$T(A \wedge B) = \min(T(A), T(B))$$
$$T(A \vee B) = \max(T(A), T(B))$$
$$T(\neg A) = 1 - T(A)$$

La lógica difusa es así un sistema de operatividad de la verdad, un hecho que causa serias dificultades. Por ejemplo, suponga que $T(Alto(Nate)) = 0{,}6$ y $T(Grueso(Nate)) = 0{,}4$. Entonces tenemos que $T(Alto(Nate) \wedge Grueso(Nate)) = 0{,}4$, que parece razonable, pero también asigna el resultado $T(Alto(Nate) \wedge \neg Alto\ (Nate)) = 0{,}4$, que no lo es. Claramente, el problema aparece de la incapacidad del enfoque de operatividad de la verdad para tener en cuenta las correlaciones o anti-correlaciones entre las proposiciones componentes.

CONTROL DIFUSO

El **control difuso** es una metodología para la construcción de sistemas de control en los que las correspondencias entre las entradas real valuadas y los parámetros de salida están representadas por reglas difusas. El control difuso ha tenido mucho éxito en productos comerciales como transmisiones automáticas, cámaras de vídeo, y máquinas de afeitar eléctricas. Los críticos (*véase*, por ejemplo, Elkan, 1993) argumentan que estas aplicaciones son exitosas porque tienen pequeñas bases de reglas, no tienen encadenamiento de inferencias, y parámetros que pueden ajustarse para mejorar el desarrollo del sistema. El hecho es que están realizados con operadores difusos que son inherentes a sus éxitos: la clave es simplemente proporcionar un modo conciso e intuitivo para especificar una función real valuada con una interpolación suave.

Ha habido intentos para dar una explicación de la lógica difusa en términos de la teoría de la probabilidad. Una idea es ver las oraciones, tales como «Nate es alta», como la realización de observaciones discretas concernientes a una variable oculta continua, la *Altura* actual de Nate. El modelo probabilista especifica $P(\text{El observador dice que Nate es alta} | Altura)$, quizás usando una **distribución probit** como se definió en el Apartado 14.3. La distribución posterior de la altura de Nate se puede entonces calcular del modo usual, por ejemplo si el modelo es parte de una red bayesiana híbrida. Tal enfoque no realiza operatividad de la verdad, por supuesto. Por ejemplo, la distribución condicionada

$$P(\text{El observador dice que Nate es alta y gruesa} | Altura, Peso)$$

tiene en cuenta las interacciones entre la altura y el peso debido a la observación. Así, a alguien que mida ocho pies y pese 190 libras es muy improbable que se le llame «alto y grueso», incluso aunque «ocho pies» se considere «alto» y «190 libras» se considere «grueso».

CONJUNTOS ALEATORIOS

Los predicados difusos también se pueden establecer dando una interpretación probabilista en términos de **conjuntos aleatorios**, esto es, variables aleatorias tales que sus valores posibles son conjuntos de objetos. Por ejemplo, *Alto* es un conjunto aleatorio cuyos valores posibles son conjuntos de personas. La probabilidad $P(Alto = S_1)$, donde S_1 es algún conjunto particular de personas, es la probabilidad de que exactamente ese conjunto estaría identificado como «alto» por el observador. Entonces la probabilidad de que «Nate es alta» es la suma de las probabilidades de todos los conjuntos en los que Nate es un miembro.

Tanto el enfoque de las redes bayesianas híbridas como el enfoque de los conjuntos aleatorios surgen para capturar aspectos relativos a lo difuso sin introducir grados de verdad. No obstante, quedan muchos tópicos abiertos referentes a la representación correcta de observaciones lingüísticas y cantidades continuas (tópicos que han sido descuidados por la mayoría que no son de la comunidad difusa).

14.8 Resumen

En este capítulo se han descrito las **redes bayesianas**, una representación bien desarrollada para el conocimiento incierto. Las redes bayesianas representan un papel casi análogo al de la lógica proposicional para conocimiento cierto.

- Una red bayesiana es un grafo acíclico dirigido cuyos nodos corresponden a variables aleatorias; cada nodo tiene una distribución condicionada para el nodo, dados sus padres.
- Las redes bayesianas proporcionan un modo conciso para representar relaciones de **independencia condicionada** del dominio.
- Una red bayesiana determina una distribución conjunta completa: cada entrada de la conjunta está definida como el producto de las correspondientes entradas de las distribuciones condicionadas locales. Una red bayesiana es a menudo exponencialmente más pequeña que la distribución conjunta completa.
- Muchas distribuciones condicionadas pueden representarse de una forma compacta usando familias canónicas de distribuciones. Las **redes bayesianas híbridas**, que incluyen tanto variables discretas como continuas, usan diversas distribuciones canónicas.
- La inferencia en redes bayesianas significa el cálculo de la distribución de probabilidad de un conjunto de variables preguntas, dado un conjunto de variables de evidencia. Los algoritmos de inferencia exacta, tal como la **eliminación de variables**, evalúa sumas de productos de probabilidades condicionadas tan eficientemente como le es posible.
- En **poliárboles** (redes con conexión simple), la inferencia exacta necesita tiempo lineal en el tamaño de los nodos. En el caso general, el problema es intratable.
- Las técnicas de aproximación estocástica tales como la **ponderación de la verosimilitud** y **Monte Carlo por cadenas de Markov** pueden proporcionar estimaciones razonables de las auténticas distribuciones posteriores de la red y pueden hacer frente a redes mucho más grandes que las que pueden tratar los algoritmos exactos.
- La teoría de la probabilidad puede combinarse con ideas de representación de la lógica de primer orden para hacer sistemas muy potentes para el razonamiento con incertidumbre. Los **modelos probabilistas relacionales** (MPRs) incluyen restricciones en la representación que garantizan una distribución de probabilidad bien definida que puede expresarse como una red bayesiana equivalente.
- Se han sugerido varios sistemas alternativos para el razonamiento con incertidumbre. Hablando en general, los sistemas con operatividad de la verdad no son muy apropiados para este tipo de razonamiento.

Notas bibliográficas e históricas

El uso de redes para representar información probabilista comenzó a principios del siglo XX, con el trabajo de Sewal Wright sobre el análisis probabilista de herencia gené-

tica y factores de crecimiento animal (Wright, 1921, 1934). Una de sus redes aparece en la cubierta de este libro. I.J. Good (1961), en colaboración con Alan Turing, desarrolló representaciones probabilistas y métodos de inferencia bayesiana que podrían considerarse como precursoras de las redes bayesianas modernas, aunque el artículo no es mucha veces citado en este contexto[14]. El mismo artículo es la fuente original para el modelo del ruido-OR.

La representación de los **diagramas de influencia** para problemas de decisión, que incorporaron una representación de GAD para las variables aleatorias, se usaron en el análisis de la decisión al final de 1970 (*véase* Capítulo 16), pero sólo la enumeración se utilizaba para la evaluación. Judea Pearl desarrolló el método de paso-de-mensajes para llevar a cabo la inferencia en redes con forma de árbol (Pearl, 1982a) y redes poliárboles (Kim y Pearl, 1983) y explicó la importancia de la construcción causal en vez de modelos probabilistas de diagnóstico, en contraste con los sistemas basados en factores de certeza entonces en boga. El primer sistema experto que usaba redes bayesianas fue CONVINCE (Kim, 1983; Kim y Pearl, 1987). Sistemas más recientes incluyen el sistema MUNIN para el diagnóstico de desórdenes neuromusculares (Andersen *et al*., 1989) y el sistema PATHFINDER para patología (Heckerman, 1991). Con mucho, los sistemas de redes bayesianas más ampliamente utilizados han sido los módulos de diagnóstico y reparación (por ejemplo, el Printer Wizard[15]) de Microsoft Windows (Breese y Heckerman, 1996) y el Office Assistant[16] de Microsoft Office (Horvitz *et al*., 1998).

Pearl (1986) desarrolló un algoritmo basado en grupos para inferencia exacta en redes bayesianas generales, utilizando una transformación a un poliárbol dirigido de grupos. Un enfoque similar, desarrollado por los estadísticos David Spiegelhalter y Steffen Lauritzen (Spiegelhalter, 1986; Lauritzen y Spiegelhalter, 1988), se basa en la transformación a una red (de Markov) no dirigida. Este enfoque está implantado en el sistema HUGIN, una herramienta eficiente y ampliamente utilizada para razonamiento incierto (Andersen *et al*., 1989). Ross Shachter, trabajando en la comunidad de los diagramas de influencia, desarrolló un método exacto basado en la reducción de la red dirigida por el objetivo, usando transformaciones que preservan el cálculo de la distribución *a posteriori* (Shachter, 1986).

El método de eliminación de variables descrito en el capítulo está más cerca en espíritu al método de Shacter, desde el cual surgió el algoritmo de inferencia probabilista simbólica (IPS) (Shachter *et al*., 1990). La IPS intenta optimizar la evaluación de expresiones en forma de árbol como el que se muestra en la Figura 14.8. El algoritmo que nosotros describimos está más cerca al desarrollado por Zhang y Poole (1994, 1996). Los criterios para podar variables irrelevantes los desarrollaron Geiger *et al*. (1990) y Lauritzen *et al*. (1990); el criterio que nosotros damos es un caso especial de éstos. Rina

---

[14] I.J. Good era jefe estadístico del equipo de descodificación de Turing en la segunda guerra mundial. En *2001: A Space Odyssey* (Clarke, 1968a), da el mérito a Good y Minsky por la innovación que condujo al desarrollo del computador HAL 9000.

[15] El término *Printer Wizard* es el nombre propio de un módulo informático, de ahí que no se traduzca. No obstante, indicar que *printer* significa impresora y *wizard* en el contexto informático viene a significar *aplicación que ayuda al usuario a ejecutar una tarea de forma eficaz*. (Nota del traductor.)

[16] El término Office Assistant es el nombre propio de un módulo informático, de ahí que no se traduzca. No obstante, indicar que vendría a significar *Ayudante de Oficina*. (Nota del traductor.)

PROGRAMACIÓN DINÁMICA NO EN SERIE

Dechter (1999) demuestra cómo la idea de eliminación de variables es esencialmente idéntica a la **programación dinámica no en serie** (Bertele y Brioschi, 1972), un enfoque algorítmico que puede aplicarse para resolver un campo de problemas de inferencia en redes bayesianas, por ejemplo, buscar la **explicación más probable** para un conjunto de observaciones. Esto conecta los algoritmos de redes bayesianas a métodos para resolver PSRs y da una medida directa de la complejidad de la inferencia exacta en términos de la **anchura del hiperárbol** de la red.

La inclusión de variables aleatorias continuas en redes bayesianas fue considerada por Pearl (1988) y Schachter y Kenley (1989); estos artículos plantearon redes que contenían sólo variables continuas con distribuciones gaussianas lineales. La inclusión de variables discretas ha sido investigada por Lauritzen y Wermuth (1989) e implantada en cHUGIN (Olesen, 1993). La distribución probit fue estudiada primero por Finney (1947), quien la llamó distribución sigmoide. Ha sido ampliamente usada para el modelado del fenómeno de la elección discreta y puede extenderse para manejar más de dos elecciones (Daganzo, 1979). Bishop (1995) da una justificación para usar la distribución logit.

Cooper (1990) demostró que el problema general de inferencia en redes bayesianas sin restricciones es NP-duro, y Paul Dagum y Mike Luby (1993) demostraron que el correspondiente problema aproximado es NP-duro. La complejidad del espacio es también un serio problema tanto en el método basado en grupos como en el de eliminación de variables. El método de **condicionamiento usando conjuntos de corte**, que fue desarrollado para PSRs en el Capítulo 5, evita la construcción de tablas exponencialmente grandes. En una red bayesiana, un conjunto de corte es un conjunto de nodos que, cuando se les asignan valores (están instanciados), reduce los nodos restantes a un poliárbol que puede resolverse en tiempo y espacio lineal. La pregunta se responde sumando en todas las instancias del conjunto de corte, así el requerimiento de espacio completo es a pesar de eso lineal (Pearl, 1988). Darwiche (2001) describe un algoritmo basado en condicionamiento que es recursivo y que permite un rango completo de posibles equilibrios entre el espacio y el tiempo empleados.

El desarrollo de algoritmos aproximados rápidos para inferencia en redes bayesianas es un área muy activa, con contribuciones de estadísticos, informáticos y físicos. El método de muestreo con rechazo es una técnica general que es conocida desde hace tiempo para los estadísticos; Max Henrion (1988) lo aplicó por primera vez a redes bayesianas, quien lo llamó **muestreo lógico**. La ponderación de la verosimilitud, que fue desarrollado por Fung y Chang (1989) y Shachter y Peot (1989), es un ejemplo del método estadístico bien conocido de **muestreo por importancia**. Una aplicación a gran escala de la ponderación de la verosimilitud a diagnóstico médico aparece en Shwe y Cooper (1991). Cheng y Druzdzel (2000) describen una versión adaptable de la ponderación de la verosimilitud que trabaja bien incluso cuando la evidencia tiene una verosimilitud *a priori* muy baja.

Los algoritmos de Monte Carlo usando cadenas de Markov (MCCM) comenzaron con el algoritmo de Metropolis, debido a Metropolis *et al*. (1953), que fue también la fuente del algoritmo de enfriamiento simulado descrito en el Capítulo 4. El muestreador de Gibbs fue diseñado por Geman y Geman (1984) para inferencia en redes de Markov no dirigidas. La aplicación de los MCCM a redes bayesianas se debe a Pearl (1987).

Los artículos recogidos por Gilks *et al.* (1996) cubren una amplia variedad de aplicaciones de los MCCM, varios de los cuales se desarrollaron en el paquete bien conocido BUGS (Gilks *et al.*, 1994).

APROXIMADO VARIABLE

Hay dos familias muy importantes de métodos aproximados que no tratamos en el capítulo. La primera es la familia de métodos **aproximados variables**, que pueden usarse para simplificar cálculos complejos de todo tipo. La idea básica es plantear una versión reducida del problema original con el que es sencillo trabajar, pero que se parezca al original tanto como sea posible. El problema reducido se describe por algunos **parámetros variables** $\boldsymbol{\lambda}$ que se ajustan para minimizar una función distancia D entre el problema original y el reducido, a menudo resolviendo el sistema de ecuaciones $\partial D/\partial \boldsymbol{\lambda} = 0$. En muchos casos, pueden obtenerse cotas superiores e inferiores estrictas. Los métodos variables han sido utilizados desde hace tiempo en estadística (Rustagi, 1976). En física estadística, el método del **campo medio** es una aproximación variable particular en el que las variables individuales que completan el modelo se supone que son completamente independientes. La idea se aplicó para resolver grandes redes de Markov no dirigidas (Peterson y Anderson, 1987; Parisi, 1988). Saul *et al.,* (1996) desarrollaron los fundamentos matemáticos para la aplicación de métodos variables a redes bayesianas y obtuvieron aproximaciones precisas de la cota inferior para redes sigmoides con el uso de métodos del campo-medio. Los enfoques variables son estudiados por Jordan *et al.* (1999).

PARÁMETRO VARIABLE

CAMPO MEDIO

Una segunda familia importante de algoritmos aproximados es la basada en el algoritmo de paso de mensajes para poliárboles de Pearl (1982a). Este algoritmo puede aplicarse a redes generales, como sugirió Pearl (1988). Los resultados pueden ser incorrectos, o el algoritmo puede fallar para terminar, pero en muchos casos, los valores obtenidos son cercanos a los valores auténticos. Poca atención se ha prestado a este enfoque conocido como **propagación de creencia** (o **propagación cíclica**) hasta que McEliece *et al.,* (1998) observaron que el paso de mensajes en una red bayesina con conexión múltiple era exactamente el cálculo que realizaba el algoritmo **turbo descodificación** (Berrou *et al.*, 1993), que proporcionó un avance importantísimo en el diseño de códigos para corrección de errores. La consecuencia es que la propagación cíclica es tanto rápida como precisa sobre las redes muy grandes y altamente relacionadas que se usan para el desciframiento y podría ser así útil en situaciones más generales. Murphy *et al.,* (1999) presentan un estudio empírico donde esto funciona realmente. Yedidia *et al.,* (2001) establecen conexiones adicionales entre la propagación cíclica e ideas de la física estadística.

PROPAGACIÓN DE CREENCIA

TURBO DESCODIFICACIÓN

La conexión entre la probabilidad y los lenguajes de primer orden la estudió primero Carnap (1950). Gaifman (1964) y Scott y Krauss (1966) definieron un lenguaje en el que las probabilidades podían asociarse con oraciones de primer orden y para el cual los modelos eran medidas de probabilidad sobre mundos posibles. Dentro de la IA, esta idea se desarrolló para la lógica proposicional por Nilsson (1986) y para la lógica de primer orden por Halpern (1990). La primera investigación amplia del tópico de la representación del conocimiento con tales lenguajes fue realizada por Bacchus (1990), y el artículo de Wellman *et al.*, (1992) estudia las primeras aproximaciones de implementación basadas en la construcción de redes bayesianas proposicionales equivalentes. Más recientemente, los investigadores han llegado a entender la importancia de las bases de

conocimiento *completas*; esto es, bases de conocimiento que, como las redes bayesianas, definen una única distribución de probabilidad conjunta sobre todos los mundos posibles. Los métodos para hacer esto se han basado en versiones probabilistas de la programación lógica (Poole, 1993; Sato y Kameya, 1997) o redes semánticas (Koller y Pfeffer, 1998). Los tipos de modelos probabilistas relacionales descritos en este capítulo se investigaron en profundida por Pfeffer (2000). Pasula y Russell (2001) estudian tanto el tópico de la incertidumbre relacional como la de la identidad dentro de los MPRs y el uso de inferencia con MCCM.

Como se explicó en el Capítulo 13, los primeros sistemas probabilistas cayeron en desgracia a principios de los 70, dejando un vacío parcial que se rellenó con métodos alternativos. Los factores de certeza se inventaron para el sistema experto médico MYCIN (Shortliffe, 1976), y que se buscó tanto como una solución de ingeniería como un modelo de juicio humano con incertidumbre. La colección *Rule-Based Expert Systems* (Buchanan y Shortliffe, 1984) proporcionan una completa visión de conjunto de MYCIN y sus descendientes (*véase* también Stefik, 1995). David Heckerman (1986) demostró que una versión levemente modificada de los cálculos de los factores de certeza consigue resultados probabilistas correctos en algunos casos, pero resultados de sobreajuste de la evidencia en otros casos. El sistema experto PROSPECTOR (Duda *et al*., 1979) usó un enfoque basado en reglas en el que las reglas se justificaban bajo supuestos (raras veces sostenibles) de independencia global.

La teoría de Dempster-Shafer se origina con el artículo de Arthur Dempster (1968) en el que se propone una generalización de la probabilidad a valores de intervalo y una regla de combinación para usarlos. El trabajo posterior de Glenn Shafer (1976) encaminó a la teoría de Dempster-Shafer a que se contemplara como un enfoque competitivo a la probabilidad. Ruspini *et al*., (1992) analizan la relación entre la teoría de Dempster-Shafer y la teoría estándar de la probabilidad. Shenoy (1989) ha propuesto un método para tomar decisiones con las funciones de creencia de Dempster-Shafer.

Los conjuntos difusos los desarrolló Lotfi Zadeh (1965) en respuesta a la dificultad detectada para suministrar entradas exactas a los sistemas inteligentes. El texto de Zimmermann (2001) proporciona una introducción detallada a la teoría de los conjuntos difusos; artículos sobre aplicaciones difusas se recogen en Zimmermann (1999). Como mencionamos en el tema, la lógica difusa a menudo se ha visto incorrectamente como un competidor directo de la teoría de la probabilidad, cuando de hecho va dirigido a un conjunto diferente de tópicos. La **teoría de la posibilidad** (Zadeh, 1978) se introdujo para tratar la incertidumbre en sistemas difusos y tiene mucho más en común con la probabilidad. Dubois y Prade (1994) dan un estudio riguroso de las conexiones entre la teoría de la posibilidad y la teoría de la probabilidad.

TEORÍA DE LA POSIBILIDAD

El renacer de la probabilidad se debe principalmente del descubrimiento de las redes bayesianas como un método para la representación y utilización de la información de independencia condicionada. Este resurgimiento no vino sin lucha; el artículo agresivo «En Defensa de la Probabilidad» (*In Defense of Probability*) de Peter Cheeseman (1985) y su posterior artículo «Una Indagación en la Comprensión de los Computadores» (*An Inquiry Into Computer Understanding*) (Cheeseman, 1988, con comentarios) da algo del sabor del debate. Una de las principales objeciones de los lógicos era que

para exigir los cálculos numéricos que la teoría de la probabilidad consideraba, no era evidente a la introspección y suponían un nivel poco realista de la exactitud de nuestro conocimiento incierto. El desarrollo de **redes probabilistas cualitativas** (Wellman, 1990a) proporcionó una abstracción puramente cualitativa de las redes bayesianas, utilizando la noción de influencias positivas y negativas entre variables. Wellman muestra que en muchos casos tal información es suficiente para toma de decisiones óptimas sin la necesidad de especificaciones exactas de los valores de probabilidad. El trabajo de Adnan Darwiche y Matt Ginsberg (1992) extraen las propiedades básicas del condicionamiento y la combinación de evidencia a partir de la teoría de la probabilidad y muestran que las propiedades pueden también aplicarse a la lógica y al razonamiento por defecto.

El sistema de tratamiento de enfermedad cardiaca descrito en el capítulo se debe a Lucas (1996). Otras aplicaciones con redes bayesianas incluyen el trabajo de Microsoft sobre la deducción de objetivos del usuario de computador a partir de sus acciones (Horvitz *et al*., 1998) y sobre filtración de correo electrónico basura (Sahami *et al*., 1998), el trabajo del Electric Power Research Institute sobre monitorización de generadores de potencia (Morjaria *et al*., 1995), y el trabajo de la NASA sobre la visualización de información de tiempo-crítico en el Mission Control de Houston (Horvitz y Barry, 1995).

Algunos de los primeros artículos importantes sobre métodos de razonamiento con incertidumbre en IA están recogidos en la antología *Readings in Uncertain Reasoning* (Shafer y Pearl, 1990) y *Uncertainty in Artificial Intelligence* (Kanal y Lemmer, 1986). La publicación más importante en el crecimiento de las redes bayesianas fue sin duda el libro *Probabilistic Reasoning in Intelligent Systems* (Pearl, 1988). Otros libros excelentes, incluyendo Lauritzen (1996), Jensen (2001) y Jordan (2003), contienen material más reciente. Nuevas investigaciones sobre razonamiento probabilista aparecen tanto en los articulos de IA de la línea central como *Artificial Intelligence* y la *Journal of AI Research*, como en artículos más especializados, tales como *International Journal of Approximate Reasoning*. Muchos artículos sobre **modelos gráficos**, que incluyen a las redes bayesianas, aparecen en revistas de estadística. Las publicaciones de las conferencias Uncertainty in Artificial Intelligence (UAI), Neural Information Processing Systems (NIPS), y Artificial Intelligence and Statistics (AISTATS) son excelentes fuentes para la investigación actual.

EJERCICIOS

14.1 Considere la red para diagnóstico de coches que se muestra en la Figura 14.18.

a) Extienda la red con las variables booleanas *TiempoHelado* y *MotorArranque*.

b) Proporcione tablas de probabilidad condicionadas razonables para todos los nodos.

c) ¿Cuántos valores independientes están incluidos en la distribución de probabilidad conjunta para los ochos nodos booleanos, suponiendo que no se conocen las relaciones de independencia condicional que se cumplen entre ellos?

Figura 14.18 Una red bayesiana que describe algunas características del sistema eléctrico y motor de un coche. Cada variable es booelena, y el valor *verdad* indica que el aspecto correspondiente del vehículo está funcionando.

d) ¿Cuántos valores de probabilidad independientes incluyen sus tablas de la red?

e) La distribución condicionada para *Arranca* podría describirse como una distribución **ruido-AND**. Defina esta familia en general y relaciónela con la distribución ruido-OR.

14.2 En la central eléctrica nuclear de su localidad hay una alarma que percibe cuándo un medidor de temperatura supera un umbral dado. El medidor cuantifica la temperatura del núcleo. Considere las variables booleanas A (la alarma suena), F_A (la alarma está defectuosa), y F_G (el medidor está defectuoso) y los nodos multivaluados G (lectura del medidor) y T (temperatura actual del núcleo).

a) Dibuje una red bayesiana para este dominio, dado que el medidor es más creíble que falle cuando la temperatura del núcleo llegue a ser demasiado alta.

b) ¿Es esta red un poliárbol?

c) Suponga que hay sólo dos posibles temperaturas actuales y medidas, normal y alta; la probabilidad de que el medidor de la temperatura correcta es x cuando está funcionando, pero y cuando está defectuoso. Proporcione la tabla de probabilidad condicionada asociada a G.

d) Suponga que la alarma funciona correctamente a no ser que esté defectuosa, en cuyo caso nunca suena. Proporcione la tabla de probabilidad condicionada asociada a A.

e) Suponga que la alarma y el medidor están funcionando y la alarma suena. Calcule una expresión para la probabilidad de que la temperatura del núcleo sea demasiado alta, en términos de las diversas probabilidades condicionadas de la red.

14.3 Dos astrónomos en diferentes partes del mundo toman las medidas M_1 y M_2 del número de estrellas N de alguna pequeña región del cielo, utilizando sus telescopios. Normalmente, hay una pequeña posibilidad e de error de hasta una estrella en cada direc-

ción. Cada telescopio puede también (con una probabilidad f mucho más pequeña) estar mal del foco (sucesos F_1 y F_2), en cuyo caso el científico descontará tres o más estrellas (o, si N es menor que tres, fracasa del todo para detectar cualquier estrella). Considere las tres redes que se muestrean en la Figura 14.19.

a) ¿Cuál de estas redes bayesianas son representaciones correctas (pero no necesariamente eficientes) de la información precedente?

b) ¿Cuál es la mejor red? Explíquelo.

c) Escriba una distribución condicionada para $\mathbf{P}(M_1|N)$, para el caso donde $N \in \{1, 2, 3\}$ y $M_1 \in \{0, 1, 2, 3, 4\}$. Cada entrada de la distribución condicionada debería expresarse como una función de los parámetros e y/o f.

d) Suponga que $M_1 = 1$ y $M_2 = 3$. ¿Cuáles son los números *posibles* de estrellas si suponemos que no hay en principio restricciones sobre los valores de N?

e) ¿Cuál es el número de estrellas *más probable*, dadas estas observaciones? Explicar cómo calcular esto, o, si no es posible calcularlo, explique qué información adicional es necesaria y cómo afectaría al resultado.

Figura 14.19 Tres posibles redes para el problema del telescopio.

14.4 Considere la red que se muestra en la Figura 14.19(ii), y suponga que los telescopios trabajan idénticamente. $N \in \{1, 2, 3\}$ y $M_1, M_2 \in \{0, 1, 2, 3, 4\}$, con las TPCs simbólicas que se definieron en el Ejercicio 14.3. Utilizando el algoritmo por enumeración, calcule la distribución de probabilidad $\mathrm{P}(N|M_1 = 2, M_2 = 2)$.

14.5 Considere la familia de redes gaussinas lineales, como se enseñó en el Apartado 14.3.

a) En una red de dos variables, sea X_1 el padre de X_2, suponga que tenemos para X una gaussiana *a priori*, y sea $\mathbf{P}(X_2| X_1)$ una distribución gaussiana lineal. Demuestre que la distribución conjunta $P(X_1, X_2)$ es una gaussiana multivarible, y calcule su matriz de covarianza.

b) Pruebe por inducción que la distribución conjunta para una red gaussiana lineal general de variables $X_1, \ldots, X_n$ es también una gaussiana multivarible.

14.6 La distribución probit definida en el Apartado 14.3 describe la distribución de probabilidad para un hijo booleano, dado un solo padre continuo.

a) ¿Cómo debe extenderse la definición para abarcar varios padres continuos?

b) ¿Cómo debe extenderse para manejar una variable hija *multivaluada*? Considere tanto los casos donde los valores del hijo están ordenados (como en la selección de una velocidad mientras conduce, dependiendo de la velocidad, desnivel, aceleración deseada, etc.) como los casos en los que están desordenados (como en la selección del autobús, tren, o coche para ir al trabajo). [*Pista*: Considere formas para dividir los valores posibles en dos conjuntos, para imitar una variable booleana.]

14.7 Este ejercicio tiene que ver con el algoritmo de eliminación de variables de la Figura 14.10.

a) La Sección 14.4 aplica la eliminación de variables a la pregunta

$$\mathbf{P}(Robo|JohnLlama = verdad, MaryLlama = verdad)$$

Desarrolle los cálculos indicados y compruebe que la respuesta es correcta.

b) Cuente el número de operaciones aritméticas realizadas, y compárelo con el número resultante de aplicar el algoritmo por enumeración.

c) Suponga que una red tiene la forma de una *cadena*: una secuencia de variables booleanas $X_1, \ldots, X_n$ donde $Padres(X_i) = \{X_{i-1}\}$ para $i = 2, \ldots, n$. ¿Cuál es la complejidad de la computación de $\mathbf{P}(X_1|X_n = verdad)$ utilizando enumeración? ¿Y usando eliminación de variables?

d) Pruebe que la complejidad en la ejecución de la eliminación de variables en redes poliárboles es lineal en el tamaño del árbol para cualquier ordenación de variables consistente con la estructura de la red.

14.8 Investigue la complejidad de la inferencia exacta en redes bayesianas generales:

a) Pruebe que cualquier problema 3-SAT puede reducirse a inferencia exacta en una red bayesiana construida para representar el problema particular, y así que la inferencia exacta es NP-duro. [*Pista:* Considere una red con una variable para cada símbolo proposición, uno para cada cláusula, y uno para la conjunción de cláusulas.]

b) El problema de contar el número de asignaciones de satisfacción para un problema 3-SAT es #P-completo. Demuestre que la inferencia exacta es al menos tan duro como éste.

14.9 Considere el problema de generación de una muestra aleatoria de una sola variable a partir de una distribución concreta. Puede suponer que un generador de números aleatorios está disponible y que devuelve un número aleatorio distribuido uniformemente entre 0 y 1.

DISTRIBUCIÓN ACUMULATIVA

a) Sea X una variable discreta con $P(X = x_i) = p_i$ para $i \in \{1, \ldots, k\}$. La **distribución acumulativa** para X da la probabilidad de que $X \in \{x_1, \ldots, x_j\}$ para cada posible j. Explique cómo calcular la distribución acumulativa en tiempo $O(k)$ y cómo generar un muestra simple de X a partir de ella. ¿Puede esto último hacerse en un tiempo menor a $O(k)$?

b) Ahora suponga que queremos generar N muestras de X, donde $N \gg k$. Explique cómo hacer esto con un tiempo esperado de ejecución por muestra que sea *constante* (es decir, independiente de k).

c) Ahora considere una variable con valores continuos con una distribución parametrizada (por ejemplo, la Gaussiana). ¿Cómo pueden generarse muestras a partir de esa distribución?

d) Suponga que quiere preguntar por una variable con valores continuos y está usando un algoritmo de muestreo como PONDERACION-VEROSIMILITUD para hacer la inferencia. ¿Cómo debería modificar el proceso de responder a preguntas?

14.10 El **manto de Markov** de una variable está definido en el Apartado 14.2.

a) Pruebe que una variable es independiente de todas las demás variables de la red, dado su manto de Markov.

b) Deduzca la ecuación (14.11).

14.11 Considere la pregunta $\mathbf{P}(Lluvia|Aspersor = verdad, HierbaHúmeda = verdad)$ para la Figura 14.11(a) y cómo el MCCM puede responderla.

a) ¿Cuántos estados tiene la cadena de Markov?

b) Calcule la **matriz de transición Q** formada por $q(\mathbf{y} \rightarrow \mathbf{y}')$ para todo $\mathbf{y}$, $\mathbf{y}'$.

c) ¿Qué representa $\mathbf{Q}^2$, la matriz de transición al cuadrado?

d) ¿Y $\mathbf{Q}^n$, cuando $n \rightarrow \infty$?

e) Explique cómo hacer inferencia probabilista en redes bayesianas, suponiendo que $\mathbf{Q}^n$ está disponible. ¿Es éste un modo práctico para hacer inferencia?

14.12 Tres equipos de fútbol *A*, *B* y *C*, juegan cada uno con el otro una sola vez. Cada partido es entre dos equipos, y puede ser ganado, empatado o perdido. Cada equipo tiene un grado desconocido y fijo de su calidad (un entero que va de 0 a 3) y el resultado del partido depende en probabilidad de la diferencia en calidad entre los dos equipos.

a) Construya un modelo probabilista relacional para describir el dominio, y proponga valores numérico para todas las distribuciones de probabilidad necesarias.

b) Construya la red bayesiana equivalente.

c) Suponga que en los primeros dos partidos *A* dé una paliza a *B* y empata con *C*. Usando un algoritmo de inferencia exacto de sus elección, calcule la distribución posterior para el resultado del tercer partido.

d) Suponga que hay *n* equipos en la liga y tenemos los resultados para todos los partidos menos el último. ¿Cómo varía la complejidad de la predicción del último juego con respecto a *n*?

e) Estudie la aplicación del MCCM a este problema. ¿Cómo de rápido converge en la práctica y cómo de bien puede escalarse?

15 Razonamiento probabilista en el tiempo

Donde intentamos interpretar el presente, entender el pasado, y quizá predecir el futuro, incluso cuando muy poco está claro cristalino.

Los agentes en entornos inciertos deben ser capaces de quedarse con la pista del estado actual del entorno, al igual que deben hacerlo los agentes lógicos. La tarea se hace más difícil por la percepción parcial y ruidosa, y la incertidumbre respecto a cómo el entorno cambia en el tiempo. En el mejor de los casos, el agente será capaz de obtener sólo una estimación probabilista de la situación actual. Este capítulo describe las representaciones y algoritmos de inferencia que hacen posible esa estimación, basados en las ideas introducidas en el Capítulo 14.

El enfoque básico se describe en el Apartado 14.1: un mundo cambiante está modelado usando una variable aleatoria para cada aspecto del estado del mundo *en cada momento en el tiempo*. Las relaciones entre estas variables definen cómo el estado evoluciona. El Apartado 15.2 define la tarea de inferencia básica y describe la estructura general de los algoritmos de inferencia para modelos temporales. Después describimos tres tipos concretos de modelos: **modelos ocultos de Markov**, **filtros de Kalman**, y **redes bayesianas dinámicas** (que incluyen a los modelos ocultos de Markov y a los filtros de Kalman como casos particulares). Finalmente, El Apartado 15.6 explica cómo los modelos probabilistas temporales forman el núcleo de los modernos sistemas de reconocimiento del habla. El aprendizaje representa un papel fundamental en la construcción de todos estos modelos, pero un estudio detallado de los algoritmos de aprendizaje se dejará para la Parte VI.

15.1 El tiempo y la incertidumbre

Hemos desarrollado nuestras técnicas para razonamiento probabilista en el contexto de mundos **estáticos**, en los que cada variable aleatoria tiene un solo valor fijo. Por ejem-

plo, cuando se repara un coche, asumimos que si está roto permanece roto durante el proceso de diagnóstico; nuestro trabajo es deducir el estado del coche a partir de la evidencia observada, que también permanece fija.

Ahora considere un problema un poco diferente: tratamiento de un paciente diabético. Como en el caso de la reparación del coche, tenemos evidencias como las dosis recientes de insulina, la comida que se ha tomado, medidas de azúcar en la sangre, y otras señales físicas. La tarea es evaluar el estado actual del paciente, incluyendo el actual nivel de azúcar en la sangre y nivel de insulina. Dada esta información, el médico (o paciente) toma una decisión sobre el consumo de comida y la dosis de insulina del paciente. A diferencia del caso de la reparación del coche, aquí los aspectos *dinámicos* del problema son esenciales. Los niveles de azúcar en la sangre y medidas de lo mismo pueden cambiar rápidamente en el tiempo, dependiendo de la comida que uno tome y la insulina que se dosifique, de la actividad metabólica de uno, el momento del día, etc. Para valorar el estado actual a partir del historial de la evidencia y para predecir las consecuencias de los tratamientos, debemos modelar estos cambios.

Las mismas consideraciones surgen en muchos otros contextos, desde el seguimiento de la actividad económica de una nación, dadas unas estadísticas parciales y aproximadas, hasta la comprensión de una secuencia de palabras habladas, dadas las mediciones acústicas ruidosas y ambiguas. ¿Cómo pueden modelarse situaciones dinámicas como estas?

Estados y observaciones

El enfoque básico que adoptaremos es similar a la idea subyacente en el cálculo de situaciones, como se explicó en el Capítulo 10: el proceso de cambio puede verse como una serie de fotos instantáneas, cada una de las cuales describe el estado del mundo en un momento particular. Cada serie de fotos instantáneas, o **cortes de tiempo**, contiene un conjunto de variables aleatorias, algunas de las cuales son observables y otras no. Por simplicidad, supondremos que el mismo subconjunto de variables es observable en cada corte (aunque esto no es estrictamente necesario en nada de lo que sigue). Usaremos $\mathbf{X}_t$ para denotar el conjunto de variables del estado no observables en el instante t y $\mathbf{E}_t$ para denotar el conjunto de variables de evidencia observables. La observación en el instante t es $\mathbf{E}_t = \mathbf{e}_t$ para algún conjunto de valores $\mathbf{e}_t$.

CORTES DE TIEMPO

Considere el siguiente ejemplo más que simplificado: suponga que es un guarda de seguridad en alguna instalación subterránea secreta. Quiere saber si está lloviendo hoy, pero su única información del mundo exterior se produce cada mañana cuando ve al director entrando con, o sin, paraguas. Para cada día t, el conjunto $\mathbf{E}_t$ contiene así una sola variable de evidencia U_t (si el paraguas está presente), y el conjunto $\mathbf{X}_t$ contiene una sola variable del estado R_t (si está lloviendo). Otros problemas pueden implicar conjuntos más grandes de variables. En el ejemplo de la diabetes, debemos tener variables de evidencias como $MedidaAzucarSangre_t$, y $NumeroPulsasiones_t$, y variables de estado como $AzucarSangre_t$ y $ContenidoEstomascal_t$[1].

[1] Nótese que $AzucarSangre_t$ y $MedidaAzucarSangre_t$ no son la misma variable; así es como nos ocupamos de las medidas ruidosas de cantidades tangibles.

El intervalo entre los cortes de tiempo también depende del problema. Para la monitorización de diabetes, un intervalo adecuado sería una hora en vez de un día. En este capítulo, supondremos generalmente un intervalo finito y fijo; esto significa que el tiempo puede etiquetarse con enteros. Supondremos que la secuencia de estados empieza en $t = 0$; por varias razones sin interés, supondremos que la evidencia empieza a llegar en $t = 1$ en vez de en $t = 0$. Así, nuestro mundo del paraguas está representado por las variables de estado $R_0, R_1, R_2, \ldots$ y las variables de evidencia $U_0, U_1, U_2, \ldots$. Usaremos la notación $a{:}b$ para denotar la secuencia de enteros de a hasta b (inclusive), y la notación $\mathbf{X}_{a:b}$ para denotar el correspondiente conjunto de variables de $\mathbf{X}_a$ hasta $\mathbf{X}_b$. Por ejemplo, $U_{1:3}$ corresponde a las variables U_1, U_2, U_3.

Procesos estacionarios e hipótesis de Markov

Con el conjunto de variables de estado y de evidencia de un problema dado, el siguiente paso es determinar las dependencias entre las variables. Podríamos seguir el procedimiento establecido en el Capítulo 14, poner las variables en algún orden y preguntar cuestiones sobre la independencia condicional de los predecesores, dado algún conjunto de padres. Una elección obvia es ordenar las variables en su orden temporal natural, ya que la causa usualmente precede al efecto y preferimos añadir las variables en el orden causal.

Sin embargo, chocaríamos rápidamente con un obstáculo: el conjunto de variables es no acotado, ya que incluye las variables de estado y de evidencia para cada corte de tiempo. Esto en realidad crea dos problemas: primero, podríamos tener que especificar un número no acotado de tablas de probabilidad condicionadas, una para cada variable de cada corte; segundo, cada una podría implicar un número no acotado de padres.

PROCESO ESTACIONARIO

El primer problema está resuelto suponiendo que los cambios del estado del mundo están causados por un **proceso estacionario**, esto es, un proceso de cambios que está controlado por leyes que no se cambian a sí mismas en el tiempo. (No confundir *estacionario* con *estático*: en un proceso *estático*, el estado en sí mismo no cambia.) En el mundo del paraguas, entonces, la probabilidad condicionada de que el paraguas está presente, $\mathbf{P}(U_t|Padres(U_t))$, es el mismo para todo t. Así, supuesto que es estacionario, necesitamos especificar las distribuciones condicionadas sólo para las variables dentro de un corte de tiempo «representativo».

HIPÓTESIS DE MARKOV

PROCESO DE MARKOV

PROCESO DE MARKOV DE PRIMER ORDEN

El segundo problema, el de la manipulación de un número de padres potencialmente infinito, se resuelve añadiendo lo que se llama **hipótesis de Markov**, esto es, que el estado actual depende sólo de un conjunto *finito* de estados precedentes. Los procesos que verifican este supuesto lo estudió primero, en profundidad, el estadístico ruso Andrei Markov y se denominan **procesos de Markov** o **cadenas de Markov**. Los hay para todos los gustos: el más simple es el **proceso de Markov de primer orden**, en el que el estado actual depende sólo del estado previo y no de los estados iniciales. En otras palabras, un estado es la información que usted necesita para hacer el futuro independiente del pasado dado el estado actual. Usando nuestra notación, la correspondiente afirmación de independencia condicionada establece que, para todo t,

$$\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1}) \tag{15.1}$$

Así, en un proceso de primer orden, las leyes que describen cómo el estado evoluciona en el tiempo están contenidas íntegramente dentro de la distribución condicionada $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$, que llamamos **modelo de transición** para procesos de primer orden[2]. El modelo de transición para un proceso de Markov de segundo orden es la distribución condicionada $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$. La Figura 15.1 muestra las estructuras de las redes bayesianas correspondientes a los procesos de primer y segundo orden.

Además de restringir a los padres de las variables de estado $\mathbf{X}_t$, debemos restringir a los padres de la variable de evidencia $\mathbf{E}_t$. Es típico suponer que las variables de evidencia en el instante t dependen sólo del estado actual:

$$\mathbf{P}(\mathbf{E}_t|\mathbf{X}_{0:t-1}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t|\mathbf{X}_t) \tag{15.2}$$

MODELO SENSOR

La distribución condicionada $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ se llama **modelo sensor** (o, en ocasiones, **modelo observación**), porque describe cómo los «sensores» (esto es, las variables de evidencia) se ven afectados por el estado actual del mundo. Note la dirección de la dependencia: la «flecha» va del estado a los valores del sensor porque el estado del mundo *causa* que los sensores tomen valores particulares. En el mundo del paraguas, por ejemplo, la lluvia *causa* que el paraguas esté presente. (El proceso de inferencia, por supuesto, va en la otra dirección; la distinción entre la dirección de las dependencias modeladas y la dirección de la inferencia es una de las principales ventajas de las redes bayesianas.)

Además del modelo transición y el modelo sensor, necesitamos especificar una probabilidad *a priori* $\mathbf{P}(\mathbf{X}_0)$ de los estados en el instante 0. Estas tres distribuciones, combinadas con las afirmaciones de independencia condicionada de las ecuaciones (15.1) y (15.2), nos dan una especificación de la distribución conjunta completa de todas las variables. Para cualquier t finita, tenemos

$$\mathbf{P}(\mathbf{X}_0, \mathbf{X}_1..., \mathbf{X}_t, \mathbf{E}_1..., \mathbf{E}_t) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i|\mathbf{X}_{i-1})\mathbf{P}(\mathbf{E}_i|\mathbf{X}_i)$$

Las suposiciones de independencia corresponden a una estructura muy sencilla de la red bayesiana que describe el sistema completo. La Figura 15.2 muestra la estructura de red para el ejemplo del paraguas, incluyendo las distribuciones condicionadas para los modelos transición y sensor.

Figura 15.1 (a) Estructura de red bayesiana correspondiente a un proceso de Markov de primer orden con el estado definido por las variables $\mathbf{X}_t$. (b) Un proceso de Markov de segundo orden.

[2] El modelo de transición es el análogo probabilista de los circuitos de actualización booleana del Capítulo 7 y los axiomas estado-sucesor del Capítulo 10.

Figura 15.2 Estructura de red bayesiana y distribuciones condicionadas describiendo el mundo del paraguas. El modelo de transición es $P(Lluvia_t|Lluvia_{t-1})$ y el modelo sensor es $P(Paraguas_t|Lluvia_t)$.

La estructura de la figura asume un proceso de Markov de primer orden, porque la probabilidad de llover se supone que depende sólo de si llovió el día previo. Si tal suposición es razonable o no depende del dominio en sí mismo. La hipótesis de Markov de primer orden dice que las variables del estado contienen *toda* la información necesaria para caracterizar la distribución de probabilidad para el siguiente corte de tiempo. Algunas veces el supuesto es realmente cierto, por ejemplo, si una partícula realiza un **camino aleatorio** a lo largo del eje x, cambiando su posición en ± 1 en cada paso a lo largo del tiempo, entonces, usando la coordenada x como el estado, se tiene un proceso de primer orden.

CAMINO ALEATORIO

A veces la hipótesis es sólo aproximada, como en el caso de predecir la lluvia sólo en base de si llovió el día anterior. Hay dos posibles ajustes si se comprueba que la aproximación es demasiado inexacta:

1. Aumentar el orden del modelo del proceso de Markov. Por ejemplo, podríamos hacer un modelo de segundo orden añadiendo $Lluvia_{t-2}$ como un padre de $Lluvia_t$, lo que debe dar un predicción un poco más adecuada (por ejemplo, en Palo Alto muy raramente llueve más de dos días seguidos).
2. Aumentar el conjunto de variables del estado. Por ejemplo, podríamos añadir $Estacion_t$ para permitirnos incorporar registros históricos de las estaciones lluviosas, o podríamos añadir $Temperatura_t$, $Humedad_t$, y $Presion_t$ para que podamos usar un modelo físico de las condiciones de las lluvias.

El Ejercicio 15.1 le pide que demuestre que la primera solución (aumentar el orden) puede siempre reformularse como un aumento del conjunto de variables del estado. Nótese que añadir variables del estado debe mejorar la potencia de predicción del sistema pero también incrementa los *requisitos* de predicción: ahora tenemos que predecir también las variables nuevas. Así, estamos buscando un conjunto de variables «autosuficiente», lo que realmente significa que tenemos que entender la «física» del proceso que se está modelando. El requerimiento para el modelado exacto del proceso puede obviamente atenuarse si podemos añadir nuevos sensores (por ejemplo, las medidas de temperatura y presión) que proporcionen información directa sobre las nuevas variables del estado.

Considere, por ejemplo, el problema de un robot que vaga de forma aleatoria en el plano X-Y. Uno podría proponer que la posición y la velocidad son un conjunto suficiente de variables del estado: uno puede simplemente usar las leyes de Newton para calcular la posición nueva, y que la velocidad puede cambiar de forma impredecible. Sin embargo, si el robot se impulsa por batería, entonces la descarga de la batería tendería a tener un efecto sistemático en el cambio de velocidad. Ya que esto a su vez depende de cuánta potencia se usó en todas las maniobras previas, la propiedad de Markov se viola. Podemos recuperar la propiedad de Markov incluyendo el nivel de carga $Bateria_t$ como una de las variables del estado que componen a $\mathbf{X}_t$. Esto ayuda en la predicción del movimiento del robot, pero a su vez requiere un modelo para la predicción de $Bateria_t$ a partir de $Bateria_{t-1}$ y la velocidad. En algunos casos, eso puede hacerse con fiabilidad; la precisión se mejoraría *añadiendo un nuevo sensor* para el nivel de batería.

15.2 Inferencia en modelos temporales

Establecida la estructura de un modelo temporal genérico, podemos formular las tareas de inferencia básica que deben resolverse.

FILTRADO

MONITORIZACIÓN

ESTADO DE CREENCIA

- **Filtrado** o **monitorización:** esta es la tarea de calcular el **estado de creencia** (la distribución *a posteriori* del estado actual, dada toda la evidencia hasta el momento). Esto es, deseamos calcular $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$, suponiendo que la evidencia aparece en un flujo continuo desde $t = 1$. En el ejemplo del paraguas, esto podría significar calcular la probabilidad de llover hoy, dadas todas las observaciones hechas hasta el momento al portador del paraguas. El filtrado es lo que un agente racional necesita hacer para mantener la pista del estado actual para que puedan tomarse decisiones racionales. (*Véase* Capítulo 17.) Resulta que un cálculo casi idéntico proporciona la **verosimilitud** de la secuencia de la evidencia $\mathbf{P}(\mathbf{e}_{1:t})$.

PREDICCIÓN

- **Predicción:** esta es la tarea de calcular la distribución *a posteriori* del estado *futuro*, dada toda la evidencia hasta el momento. Esto es, deseamos calcular $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ para algún $k > 0$. En el ejemplo del paraguas, esto podría significar calcular la probabilidad de llover tres días a partir de ahora, dadas todas las observaciones hechas hasta el momento al portador del paraguas. La predicción es útil para la evaluación de los posibles cursos de acción.

SUAVIZADO

RETROSPECTIVA

- **Suavizado,** o **retrospectiva**: es la tarea de calcular la distribución *a posteriori* del estado *pasado*, dada toda la evidencia hasta el momento presente. Esto es, deseamos calcular $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ para algún k tal que $0 \leqslant k < t$. En el ejemplo del paraguas, podría significar calcular la probabilidad de que llovió el último miércoles, dadas todas las observaciones hechas hasta hoy al portador del paraguas. La retrospectiva proporciona un estimación mejor del estado que estaba disponible en el momento, porque el estado incorpora más evidencia.
- **Explicación más creíble:** dada una secuencia de observaciones, podemos desear encontrar la secuencia de estados más creíbles que hayan generado esas observaciones. Esto es, deseamos calcular $\text{argmax}_{\mathbf{X}_{1:t}} \mathbf{P}(\mathbf{X}_{1:t}|\mathbf{e}_{1:t})$. Por ejemplo, si el paraguas está presente en cada uno de los tres primeros días y está ausente en el cuarto,

entonces la explicación más probable es que llovió los tres primeros días y no llovió el cuarto. Los algoritmos para esta tarea son útiles en muchas explicaciones, incluyendo el reconocimiento del habla (donde la finalidad es encontrar la secuencia más probable de palabras, dada una serie de sonidos) y la reconstrucción de una serie de bits transmitidos por un canal ruidoso.

Además de estas tareas, los métodos son necesarios para el *aprendizaje* de los modelos de transición y sensor a partir de las observaciones. Al igual que con las redes bayesianas estáticas, el aprendizaje de redes de Bayes dinámicas puede hacerse como el resultado de un producto de inferencias. La inferencia proporciona una estimación de qué transiciones realmente ocurrieron y de qué estados generaron la lectura de los sensores, y estas estimaciones pueden usarse para actualizar el modelo. Los modelos actualizados proporcionan nuevas estimaciones, y el proceso itera hasta converger. El proceso global es un caso del **algoritmo EM** o expectación-maximización. (*Véase* Apartado 20.3.) Un punto a notar es que el aprendizaje requiere la inferencia de suavizado completa, en vez de con filtrado, porque proporciona mejores estimaciones de los estados del proceso. El aprendizaje con filtrado puede fracasar en la convergencia correcta; considere, por ejemplo, el problema del aprendizaje para resolver homicidios: la restropectiva *siempre* es necesaria para inferir qué ocurrió en la escena del homicidio a partir de las variables observables.

Los algoritmos para las cuatro tareas de inferencia enumeradas en el párrafo anterior pueden describirse primero a un nivel genérico, independientemente del tipo particular de modelo empleado. En las secciones correspondientes, se describirán mejoras específicas de cada modelo.

Filtrado y predicción

Comencemos con el filtrado. Mostraremos que esto puede hacerse a un estilo *online* sencillo: dado el resultado del filtrado en el instante t, uno puede calcular fácilmente el resultado para $t + 1$ a partir de la evidencia $\mathbf{e}_{t+1}$. Esto es,

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t}))$$

ESTIMACIÓN RECURSIVA

para alguna función f. Este proceso a menudo se llama **estimación recursiva**. Podemos ver el cálculo como el compuesto realmente de dos partes: primero, la distribución del estado actual se proyecta desde t hasta $t + 1$; entonces se actualiza utilizando la nueva evidencia $\mathbf{e}_{t+1}$. Este proceso de dos partes resulta realmente simple:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \quad \text{(dividiendo la evidencia)}\\ &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t})\, \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) \quad \text{(usando la regla de Bayes)}\\ &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\, \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) \quad \text{(por la propiedad de Markov en la evidencia)}\end{aligned}$$

Aquí y en todo este capítulo, α es una constante de normalización utilizada para hacer que las probabilidades sumen 1. El segundo término, $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$, representa una predicción en un paso para el estado siguiente y el primer término actualiza esto con la nueva evidencia; nótese que $\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})$ puede obtenerse directamente a partir del modelo

sensor. Ahora, obtendremos la predicción en un paso para el estado siguiente condicionando al estado actual $\mathbf{X}_t$:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)\, P(\mathbf{x}_t|\mathbf{e}_{1:t}) \text{ (usando la propiedad de Markov)}\end{aligned} \tag{15.3}$$

En la sumatoria, el primer factor es sencillamente el modelo de transición y el segundo es la distribución del estado actual. Así, tenemos la fórmula recursiva deseada. Podemos pensar en la estimación filtrada $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ como un «mensaje» $\mathbf{f}_{1:t}$ que se propaga hacia delante a lo largo de la secuencia, modificado por cada transición y actualizado por cada observación nueva. El proceso es:

$$\mathbf{f}_{1:t+1} = \alpha\, \textsc{Hacia-Adelante}\,(\mathbf{f}_{1:t}\,, \mathbf{e}_{t+1})$$

donde HACIA-ADELANTE pone en marcha la actualización descrita en la Ecuación (15.3).

Cuando todas las variables del estado son discretas, el tiempo para cada actualización es constante (es decir, independiente de t), y el espacio requerido es también constante. (Las constantes dependen, por supuesto, del tamaño del espacio de estados y el tipo concreto de modelo temporal en cuestión.) *Los requerimientos de tiempo y espacio para la actualización tienen que ser constantes si un agente con memoria limitada mantiene la pista de la distribución del estado actual a través de una secuencia no acotada de observaciones.*

Ilustramos el proceso de filtrado para dos pasos en el ejemplo base del paraguas. (*Véase* Figura 15.2.) Supongamos que nuestro guardia de seguridad tiene alguna creencia *a priori* sobre si llovió el día 0, justo antes de que comience la secuencia de observaciones. Supongamos que ésta es $\mathbf{P}(R_0) = \langle 0{,}5, 0{,}5\rangle$. Ahora procesamos las dos observaciones como sigue:

- En el día 1, el paraguas aparece, así $U_1 =$ *verdad*. La predicción desde $t = 0$ hasta $t = 1$ es

  $$\mathbf{P}(R_1) = \sum_{r_0} \mathbf{P}(R_1|r_0)P(r_0)P(R_1) = \langle 0{,}7, 0{,}3\rangle \times 0{,}5 + \langle 0{,}3, 0{,}7\rangle \times 0{,}5 = \langle 0{,}5, 0{,}5\rangle,$$

  y actualizándola con la evidencia para $t = 1$ produce

  $$\begin{aligned}\mathbf{P}(R_1|U_1) &= \alpha\, \mathbf{P}(U_1|R_1)\mathbf{P}(R_1) = \alpha\, \langle 0{,}9, 0{,}2\rangle\, \langle 0{,}5, 0{,}5\rangle \\ &= \alpha\, \langle 0{,}45, 0{,}1\rangle \approx \langle 0{,}818, 0{,}182\rangle\end{aligned}$$

- En el día 2, el paraguas aparece, así $U_2 =$ *verdad*. La predicción desde $t = 1$ hasta $t = 2$ es

  $$\begin{aligned}\mathbf{P}(R_2|U_1) = \sum_{r_1} \mathbf{P}(R_2|r_1)\mathbf{P}(r_1|U_1)\mathbf{P}(R_2) &= \langle 0{,}7, 0{,}3\rangle \times 0{,}818 + \langle 0{,}3, 0{,}7\rangle \\ &\quad \times 0{,}182 \approx \langle 0{,}627, 0{,}373\rangle,\end{aligned}$$

  y actualizándola con la evidencia para $t = 2$ produce

  $$\begin{aligned}\mathbf{P}(R_2|U_1, U_2) &= \alpha\, \mathbf{P}(U_2|R_2)\mathbf{P}(R_2|U_1) = \alpha\, \langle 0{,}9, 0{,}2\rangle\, \langle 0{,}627, 0{,}373\rangle \\ &= \alpha\, \langle 0{,}565, 0{,}075\rangle \approx \langle 0{,}883, 0{,}117\rangle\end{aligned}$$

Intuitivamente, la probabilidad de lluvia aumenta del día 1 al día 2 porque la lluvia persiste. El Ejercicio 15.2(a) le pide que estudie esta tendencia más a fondo.

La tarea de la **predicción** puede verse sencillamente como un filtrado pero sin añadir la nueva evidencia. De hecho, el proceso de filtrado ya incorpora una predicción de un paso, y es fácil deducir el siguiente cómputo recursivo para la predicción del estado en $t + k + 1$ a partir de una predicción para $t + k$:

$$\mathbf{P}(\mathbf{X}_{t+k+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{X}_{t+k+1}|\mathbf{x}_{t+k})P(\mathbf{x}_{t+k}|\mathbf{e}_{1:t}) \tag{15.4}$$

Naturalmente, este cálculo involucra sólo el modelo de transición y no al modelo sensor.

Es interesante considerar qué ocurre cuando intentamos predecir más y más aún en el futuro. Como muestra el Ejercicio 15.2(b), la distribución predictiva para la lluvia converge a un punto fijo $\langle 0{,}5,\ 0{,}5\rangle$, después del cual permanece constante todo el tiempo. Esta es la **distribución estacionaria** del proceso de Markov definida por el modelo de transición. (*Véase* también Apartado 14.5.) Se conoce un gran problema sobre las propiedades de tales distribuciones y sobre el **tiempo de mezcla** (en líneas generales, el tiempo necesario para alcanzar el punto fijo). En términos prácticos, esto conduce al fracaso de cualquier intento de predecir el estado *actual* para un número de pasos que sea mayor que una fracción pequeña del tiempo de mezcla. Cuanta más incertidumbre haya en el modelo de transición, más corto será el tiempo de mezcla y más confuso el futuro.

TIEMPO DE MEZCLA

Además del filtrado y la predicción, podemos usar un método recursivo hacia delante para calcular la **verosimilitud** de la secuencia de la evidencia, $P(\mathbf{e}_{1:t})$. Ésta es una cantidad útil si queremos comparar diferentes modelos temporales que puedan producir la misma secuencia de evidencia; por ejemplo, en el Apartado 15.6, comparamos diferentes palabras que pueden producir la misma secuencia de sonido. Para esta recursión, usamos un mensaje de verosimilitud $\ell_{1:t}=\mathbf{P}(\mathbf{X}_t,\mathbf{e}_{1:t})$. Es un ejercicio sencillo demostrar que

$$\ell_{1:t+1} = \textsc{Hacia-Adelante}\ (\ell_{1:t}, \mathbf{e}_{t+1})$$

Calculada $\ell_{1:t}$, obtenemos la versosimilitud real sumando en $\mathbf{X}_t$:

$$\ell_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \ell_{1:t}(\mathbf{x}_t) \tag{15.5}$$

Suavizado

Como dijimos anteriormente, el suavizado es el proceso de calcular la distribución de los estados pasados dada la evidencia hasta el estado presente; esto es, $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ para $1 \leqslant k < t$. (*Véase* Figura 15.3.) Esto se hace por conveniencia en dos partes, considerando la evidencia hasta k y la evidencia desde $k + 1$ hasta t,

$$\begin{aligned}
\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k,\mathbf{e}_{1:k}) \quad \text{(usando la regla de Bayes)} \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \quad \text{(usando independencia condicional)} \\
&= \alpha\, \mathbf{f}_{1:k}\mathbf{b}_{k+1:t},
\end{aligned} \tag{15.6}$$

donde hemos definido un mensaje «hacia atrás» $\mathbf{b}_{k+1:t} = \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k)$, análogo al mensaje hacia delante $\mathbf{f}_{1:k}$. El mensajes hacia delante $\mathbf{f}_{1:k}$ puede calcularse por filtrado hacia delante de 1 hasta k, como viene en la Ecuación (15.3). Resulta que el mensaje hacia

Figura 15.3 El suavizado calcula $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$, la distribución posteriori del estado en algún momento pasado k dada un secuencia completa de observaciones desde 1 hasta t.

atrás $\mathbf{b}_{k+1:t}$ puede calcularse por un proceso recursivo que se ejecuta *hacia atrás* a partir de t:

$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \quad \text{(condicionando sobre } \mathbf{X}_{k+1}\text{)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \quad \text{(por independencia condicional)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k), \qquad (15.7)
\end{aligned}$$

donde el último paso se obtiene por la independencia condicional de $\mathbf{e}_{k+1}$ y $\mathbf{e}_{k+2:t}$, dada $\mathbf{X}_{k+1}$. De los tres factores de la sumatoria, el primero y el tercero se obtienen directamente del modelo, y el segundo es la «llamada recursiva». Usando la notación de mensajes, tenemos

$$\mathbf{b}_{k+1:t} = \text{HACIA-ATRÁS}\ (\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$$

donde HACIA-ATRÁS implementa la actualización descrita en la Ecuación (15.7). Como con el recursividad hacia delante, el tiempo y el espacio necesarios para cada actualización son constantes y así independientes de t.

Ahora podemos ver que los dos términos de la Ecuación (15.6) pueden calcularse por recursividad a lo largo del tiempo, uno se ejecuta hacia delante desde 1 hasta k usando la ecuación de filtrado (15.3) y el otro se ejecuta hacia atrás desde t hasta $k + 1$ y usando la Ecuación (15.7). Nótese que la fase hacia atrás está inicializada con $\mathbf{b}_{t+1:t} = \mathbf{P}(\mathbf{e}_{t+1:t}|\mathbf{X}_t) = \mathbf{1}$, donde $\mathbf{1}$ es un vector de unos. (Ya que $\mathbf{e}_{t+1:t}$ es una secuencia vacía, la probabilidad de acecho es 1.)

Apliquemos ahora este algoritmo al ejemplo del paraguas, calculada la estimación suavizada de la probabilidad de que llueva en $t = 1$, dadas las observaciones del paraguas en los días 1 y 2. De la Ecuación (15.6), esto viene dado por

$$\mathbf{P}(R_1|U_1, U_2) = \alpha\, \mathbf{P}(R_1|U_1)\mathbf{P}(U_2|R_1) \qquad (15.8)$$

El primer término ya sabemos que es $\langle ,818, ,182\rangle$, obtenido por el proceso de filtrado hacia delante descrito anteriormente. El segundo términos puede calcularse aplicando la recursividad hacia atrás de la Ecuación (15.7):

$$\begin{aligned}
\mathbf{P}(U_2|R_1) = \sum_{r_2} P(U_2|r_2)P(\ |r_2)\mathbf{P}(r_2|R_1) &= (0{,}9 \times 1 \times \langle 0{,}7, 0{,}3\rangle) \\
&+ (0{,}2 \times 1 \times \langle 0{,}3, 0{,}7\rangle) = \langle 0{,}69, 0{,}41\rangle
\end{aligned}$$

Colocando esto en la ecuación (15.8), hallamos que la estimación suavizada para la lluvia en el día 1 es

$$\mathbf{P}(R_1|U_1, U_2) = \alpha \langle 0{,}818, 0{,}182\rangle \times \langle 0{,}69, 0{,}41\rangle \approx \langle 0{,}883, 0{,}117\rangle$$

Así, la estimación suavizada es *mayor* que la estimación filtrada $\langle 0{,}818\rangle$ en este caso. Esto es porque el paraguas en el día 2 hace más creíble que haya llovido en el día 2; a su vez, ya que la lluvia tiende a persistir, eso hace más creíble que haya llovido en el día 1.

Tanto las recursividades hacia delante y hacia atrás necesitan una cantidad constante de tiempo por paso; así, el tiempo de complejidad del suavizado con respecto a la evidencia $\mathbf{e}_{1:t}$ es $O(t)$. Esta es la complejidad para el suavizado en un paso de tiempo concreto k. Si queremos suavizar la secuencia completa, un método obvio es simplemente ejecutar el proceso de suavizado completo una vez para cada paso de tiempo que va a ser suavizado. Esto tiene como resultado una complejidad de $O(t^2)$. Una estrategia mejor usa una aplicación muy simple de la programación dinámica para reducir la complejidad a $O(t)$. Una pista aparece en el análisis precedente del ejemplo del paraguas, donde éramos capaces de reutilizar el resultado de la fase de filtrado hacia delante. La clave para el algoritmo de tiempo-lineal es *grabar el resultado* del filtrado hacia delante sobre la secuencia completa. Entonces ejecutamos la recursividad hacia atrás desde t retrocediendo hasta 1, calculando la estimación suavizada en cada paso k desde el mensaje hacia atrás calculado $\mathbf{b}_{k+1:t}$ y el mensaje hacia delante almacenado $\mathbf{f}_{1:k}$. El algoritmo, adecuadamente llamado **algoritmo hacia delante-atrás**, se muestra en la Figura 15.4.

ALGORITMO HACIA DELANTE-ATRÁS

El lector avezado habrá descubierto que la estructura de la red bayesiana que se muestra en la Figura 15.3 es un **poliárbol** según la terminología del Capítulo 14. Esto significa que una aplicación sencilla del algoritmo basado en grupos también conduce a un algoritmo de tiempo-lineal que calcula estimaciones suavizadas para la secuencia entera. El algoritmo hacia delante-atrás es de hecho un caso particular del algoritmo de pro-

```
función HACIA DELANTE-ATRÁS (ev, priori) devuelve un vector de distribuciones de probabilidad.
  entradas: ev, un vector de valores de evidencia para los pasos 1,…,t
            priori, la distribución a priori sobre los estados iniciales, P(X0)
  variables locales: fv, un vector de mensajes hacia delante para los pasos 0,…,t
                     b, una representación del mensaje hacia atrás, inicialmente todos 1s
                     sv, un vector de estimaciones suavizadas para los pasos 1,…,t

  fv[0] ← priori
  desde i = 1 hasta t hacer
    fv[i] ← HACIA-DELANTE (fv[i-1],ev[i])
  desde i = t retrocediendo hasta 1 hacer
    sv[i] ← NORMALIZAR (fv[i] × b)
    b ← HACIA-ATRAS (b,ev[i])
  retorna sv
```

Figura 15.4 El algoritmo hacia delante-atrás para calcular probabilidades posteriores de una secuencia de estados dada una secuencia de observaciones. Los operadores HACIA-DELANTE y HACIA-ATRÁS están definidos por las Ecuaciones (15.3) y (15.7), respectivamente.

pagación de poliárboles usado con los métodos basados en grupos (aunque los dos fueron desarrollados independientemente).

El algoritmo hacia delante-atrás constituye la columna vertebral de los métodos computacionales empleados en muchas aplicaciones que tratan con secuencias de observaciones ruidosas, yendo desde el reconocimiento del habla hasta el rastreo de radares de aviones. Como señalamos, tiene dos desventajas prácticas. La primera es que su complejidad en espacio puede ser demasiado alta para aplicaciones donde el espacio de estados es grande y las secuencias son largas. Usa espacio $O(|\mathbf{f}|t)$ donde $|\mathbf{f}|$ es el tamaño de la representación del mensaje hacia delante. El requerimiento de espacio puede reducirse a $O(|\mathbf{f}| \log t)$ con un incremento simultáneo en la complejidad temporal por un factor de $\log t$, como se muestra en el Ejercicio 15.3. En algunos casos (*véase* Apartado 15.3), un algoritmo de espacio-constante puede usarse sin penalización en el tiempo.

SUAVIZADO DE LAPSO-FIJO

El segundo inconveniente del algoritmo básico es que necesita modificarse para trabajar en un escenario *online* donde las estimaciones suavizadas deben calcularse para cortes de tiempo iniciales conforme se van añadiendo nuevas observaciones al final de la secuencia. La necesidad más común es la del **suavizado de lapso-fijo**, que necesita calcular la estimación suavizada $\mathbf{P}(\mathbf{X}_{t-d}|\mathbf{e}_{1:t})$ para un d fijado. Esto es, el suavizado se hace para los d pasos de corte de tiempo comenzando desde el tiempo actual t; conforme t se incrementa, el suavizado tiene que mantenerse. Obviamente, podemos ejecutar el algoritmo hacia delante-atrás sobre la «ventana» de d-pasos conforme se añada cada nueva observación, pero esto es ineficiente. En el Apartado 15.3, veremos que el suavizado de lapso-fijo puede, en algunos casos, hacerse en tiempo constante por actualización, independientemente del lapso d.

Encontrar la secuencia más probable

Suponga que [*verdad, verdad, falso, verdad, verdad*] es la secuencia de las observaciones del paraguas para los primeros cinco días de trabajo para el guardia de seguridad. ¿Cuál es la secuencia del tiempo climatológico más probable que explique esto? ¿La ausencia del paraguas el día 3 significa que no estaba lloviendo, o el director olvidó traerlo? Si no llovió el tercer día, quizá (como el tiempo tiende a persistir) no llovió el día 4 tampoco, pero el director trajo el paraguas por si acaso. En total hay 2^5 posibles secuencias de tiempo que podríamos escoger. ¿Hay un modo de encontrar la más probable que no sea enumerarlas todas?

Una aproximación que podríamos intentar es el siguiente procedimiento de tiempo-lineal: usar el algoritmo de suavizado para encontrar la distribución a posteriori para el tiempo climático en cada paso de tiempo; entonces construir la secuencia, usando en cada paso el tiempo climático más probable atendiendo a la posteriori. Tal aproximación debería apagar las campanas de alarma en la cabeza del lector ya que las *a posteriori* calculadas por suavizado son distribuciones sobre pasos de tiempo *individuales*, mientras que para encontrar la *secuencia* más probable debemos considerar probabilidades *conjuntas* sobre todos los pasos de tiempo. Los resultados pueden de hecho ser completamente diferentes. (*Véase* Ejercicio 15.4).

Hay un algoritmo de tiempo-lineal para encontrar la secuencia más probable, pero requiere unas pocas ideas más. Se basa en la misma propiedad de Markov que condujeron a los algoritmos eficientes para el filtrado y el suavizado. El modo más sencillo para pensar en el problema es ver cada secuencia como un *camino* a través de un grafo cuyos nodos

son los posibles *estados* en cada paso de tiempo. Tal grafo para el mundo del paraguas se muestra en la Figura 15.5(a). Ahora considere la tarea de encontrar el camino más probable a través del grafo, donde la verosimilitud de cualquier camino es el producto de las probabilidades de transición a lo largo del camino y las probabilidades de las observaciones dadas en cada estado. Nos centramos en caminos que alcancen el estado $Lluvia_5 = verdad$. Por la propiedad de Markov, se sigue que el camino más probable hasta el estado $Lluvia_5 = verdad$ consta del camino más probable para algún estado en el instante 4 seguido por una transición a $Lluvia_5 = verdad$; y el estado en el instante 4 que llegue a formar parte del camino hasta $Lluvia_5 = verdad$ es cualquiera que maximice la verosimilitud de ese camino. En otras palabras, *hay una relación recursiva entre los caminos más probables para cada estado* $\mathbf{x}_{t+1}$ *y los caminos más probables para cada estado* $\mathbf{x}_t$. Podemos escribir esta relación como una ecuación que conecta las probabilidades de los caminos:

$$\max_{\mathbf{x}_1 \ldots \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1,\ldots,\mathbf{x}_t, \mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) =$$
$$= \alpha\, \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \left(\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t) \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} \mathrm{P}(\mathbf{x}_1,\ldots,\mathbf{x}_{t+1}, \mathbf{x}_t|\mathbf{e}_{1:t}) \right) \qquad (15.9)$$

La Ecuación (15.9) es *idéntica* a la ecuación de filtrado (15.3) excepto que

1. El mensaje hacia delante $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t}$ se reemplaza por el mensaje
$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} \mathrm{P}(\mathbf{x}_1,\ldots,\mathbf{x}_{t+1}, \mathbf{X}_t|\mathbf{e}_{1:t})$$
esto es, las probabilidades del camino más probable para cada estado $\mathbf{x}_t$; y.
2. La sumatoria sobre $\mathbf{x}_t$ en la Ecuación (15.3) se reemplaza por la maximización sobre $\mathbf{x}_t$ en la Ecuación (15.9).

Figura 15.5 (a) Posibles secuencias de estados para $Lluvia_t$ pueden verse como caminos a través de un grafo con posibles estados en cada paso de tiempo. (Los estados se muestran como nodos cuadrados para evitar confusión con los nodos de la red bayesiana.) (b) Las operaciones del algoritmo de Viterbi para la secuencia de observaciones del paraguas [*verdad, verdad, falso, verdad, verdad*]. Para cada paso de tiempo t, mostramos los valores del mensaje $\mathbf{m}_{1:t}$, que da la probabilidad de la mejor secuencia que alcanza a cada estado en el instante t. Además, para cada estado, la flecha en negrita que llega a él indica su mejor predecesor. Siguiendo las flechas en negrita en orden inverso desde el estado $\mathbf{m}_{1:5}$ se obtiene la secuencia más probable.

Así, el algoritmo para calcular la secuencia más probable es similar al filtrado: se ejecuta hacia delante a lo largo de la secuencia, calculando el mensaje **m** en cada paso de tiempo, usando la Ecuación (15.9). El progreso de este cálculo se muestra en la Figura 15.5(b). Al final, se tendrá la probabilidad de la secuencia más probable que alcance *cada uno* de los estados finales. Uno puede así fácilmente seleccionar la secuencia global más probable (el estado trazado en negrita). Para identificar la secuencia real, a diferencia de calcular su probabilidad, el algoritmo también necesita mantener punteros desde cada estado al mejor estado que lo produjo (mostrado en negrita); la secuencia se identifica siguiendo los punteros (en dirección inversa) desde el mejor estado final.

ALGORITMO DE VITERBI

El algoritmo que acabamos de describir se llama **algoritmo de Viterbi**, después de su inventor. Como el algoritmo de filtrado, su complejidad es lineal en t, la longitud de la secuencia. Sin embargo, a diferencia del filtrado su requerimiento de espacio es también lineal en t. Esto es porque el algoritmo de Viterbi necesita mantener los punteros que identifican la mejor secuencia que conducen a cada estado.

15.3 Modelos ocultos de Markov

En la sección anterior se desarrollaron algoritmos para el razonamiento probabilista temporal usando un marco general que era independiente de las expresiones concretas de los modelos de transición y sensor. En esta sección y en las dos siguientes, discutimos modelos más concretos y aplicaciones que ilustran la potencia de los algoritmos básicos y en algunos casos permite más mejoras.

MODELO OCULTO DE MARKOV

Comenzamos con el **modelo oculto de Markov**, o **MOM**. Un MOM es un modelo probabilista temporal en el que cada estado del proceso está definido por una *única* variable aleatoria *discreta*. Los valores posibles de la variable son los estados posibles del mundo. El ejemplo del paraguas descrito en la sección anterior es así un MOM, ya que tiene una variable estado: $Lluvia_t$. Se pueden añadir variables de estado adicionales a un modelo temporal manteniéndonos en el marco del MOM, pero sólo si combinamos todas las variables de estado en un sola «megavariable» cuyos valores son todas las posibles tuplas de valores de las variables de estado individuales. Veremos que la estructura limitada de los MOMs permite una implementación matricial elegante y sencilla de todos los algoritmos básicos[3]. El Apartado 15.6 muestra cómo los MOMs se utilizan para el reconocimiento del habla.

Algoritmos matriciales simplificados

Con una sola variable de estado discreta X_t podemos dar expresiones concretas a las representaciones del modelo de transición, el modelos sensor, y los mensajes hacia delante y hacia atrás. Consideremos que la variable de estado X_t tiene valores denotados por en-

[3] El lector no familiarizado con las operaciones básicas con vectores y matrices podría desear consultar el Apéndice A antes de proseguir con esta sección.

teros 1,…,S, donde S es el número de posibles estados. El modelo de transición $\mathbf{P}(X_t|X_{t-1})$ se convierte en una matriz $\mathbf{T}$ de dimensión $S \times S$, donde

$$\mathbf{T}_{ij} = P(X_t = j|X_{t-1} = i)$$

Esto es, $\mathbf{T}_{ij}$ es la probabilidad de una transición desde el estado i hasta el estado j. Por ejemplo, la matriz de transición para el mundo del paraguas es

$$\mathbf{T} = \mathbf{P}(X_t|X_{t-1}) = \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{pmatrix}$$

También ponemos el modelo sensor en forma matricial. En este caso, ya que el valor de la variable evidencia E_t es conocida, digamos, e_t, necesitamos usar sólo esa parte del modelo que especifica la probabilidad de que e_t aparezca. Para cada paso de tiempo t, construimos una matriz diagonal $\mathbf{O}_t$ cuyas entradas diagonales están dadas por los valores $P(e_t|X_t = i)$ y el resto de las entradas son 0. Por ejemplo, en el día 1 del mundo del paraguas, $U_1 = verdad$, así, de la Figura 15.2, tenemos

$$\mathbf{O}_1 = \begin{pmatrix} 0{,}9 & 0 \\ 0 & 0{,}2 \end{pmatrix}$$

Ahora, si usamos vectores columna para representar los mensajes hacia delante y hacia atrás, los cálculos se convierten en sencillas operaciones matriciales-vectoriales. La ecuación hacia delante (15.3) se transforma en:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \tag{15.10}$$

y la ecuación hacia atrás (15.7) se convierte en

$$\mathbf{b}_{k+1:t} = \mathbf{T}\mathbf{O}_{k+1}\mathbf{b}_{k+2:t} \tag{15.11}$$

De estas ecuaciones, podemos ver que la complejidad en tiempo del algoritmo hacia delante-atrás (Figura 15.4) aplicado a una secuencia de longitud t es $O(S^2t)$, ya que cada paso requiere multiplicar un vector de S-elementos por una matriz $S \times S$. El requerimiento de espacio es $O(St)$, ya que los pasos hacia delante almacenan t vectores de tamaño S.

Además proporciona una descripción elegante de los algoritmos de filtrado y suavizado para MOMs, la formulación matricial revela posibilidades para mejorar los algoritmos. La primera es una variación sencilla en el algoritmo hacia delante-atrás que permite que el suavizado se lleve a cabo en un espacio *constante*, independientemente de la longitud de la secuencia. La idea es que el suavizado para cualquier corte de tiempo concreto k requiere la presencia simultánea de tanto los mensajes hacia delante como de los de hacia atrás, $\mathbf{f}_{1:k}$ y $\mathbf{b}_{k+1:t}$, de acuerdo a la Ecuación (15.6). El algoritmo hacia delante-atrás consigue esto almacenando los **f**s calculados en el paso hacia delante con el fin de que estén disponibles durante el paso hacia atrás. Otro modo de lograr esto es con un paso sencillo que propague tanto **f** y **b** en la misma dirección. Por ejemplo, el mensaje «hacia delante» **f** puede propagarse hacia atrás si manipulamos la Ecuación (15.10) para que trabaje en la otra dirección:

$$\mathbf{f}_{1:t} = \alpha'(\mathbf{T}^\top)^{-1}\mathbf{O}_{t+1}^{-1}\,\mathbf{f}_{1:t+1}$$

El algoritmo de suavizado modificado trabaja ejecutando primero el paso hacia delante estándar para calcular $\mathbf{f}_{t:t}$ (olvidando todos los resultados intermedios) y entonces eje-

cutando el paso hacia atrás para tanto **b** y **f** juntos, usándolos para calcular la estimación suavizada en cada paso. Como sólo es necesario una copia de cada mensaje, los requerimientos de almacenamiento son constantes (es decir, independiente de t, la longitud de la secuencia). Hay una restricción crítica en este algoritmo: requiere que la matriz de transición se pueda invertir y que el modelo sensor no tenga ceros, esto es, cada observación es posible en cada estado.

Una segunda área en la que la formulación matricial revela una mejora es en el suavizado *online* con un lapso fijo. El hecho de que el suavizado puede hacerse en espacio constante sugiere que debería existir un algoritmo recursivo eficiente para suavizado *online* (esto es, un algoritmo cuyo tiempo de complejidad es independiente de la longitud del lapso). Supongamos que el lapso es d; esto es, estamos suavizando en el corte de tiempo t-d, donde el instante actual es t. Por la Ecuación (15.6), necesitamos calcular

$$\alpha \mathbf{f}_{1:t-d}\mathbf{b}_{t-d+1:t}$$

para el corte $t - d$. Entonces, cuando viene una nueva observación, necesitamos calcular

$$\alpha \mathbf{f}_{1:t-d+1}\mathbf{b}_{t-d+2:t+1}$$

para el corte $t - d + 1$. ¿Cómo podemos hacer esto incrementalmente? Primero, podemos calcular $\mathbf{f}_{1:t-d+1}$ a partir de $\mathbf{f}_{1:t-d}$, utilizando el proceso de filtrado estándar, Ecuación (15.3).

El cálculo del mensaje hacia atrás incrementalmente es más difícil, ya que no hay una relación sencilla entre el viejo mensaje hacia atrás $\mathbf{b}_{t-d+1:t}$ y el nuevo mensaje hacia atrás $\mathbf{b}_{t-d+2:t+1}$. Así, examinaremos la relación entre el viejo mensaje hacia atrás $\mathbf{b}_{t-d+1:t}$ y el mensaje hacia atrás al principio de la secuencia, $\mathbf{b}_{t+1:t}$. Para hacer esto, aplicamos la Ecuación (15.11) d veces para obtener

$$\mathbf{b}_{t-d+1:t} = \left(\prod_{i=t-d+1}^{t} \mathbf{TO}_i \right) \mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t}\mathbf{1}, \tag{15.12}$$

donde la matríz $\mathbf{B}_{t-d+1:t}$ es el producto de la secuencia de las matrices **T** y **O**. **B** puede pensarse como un «operador de transformación» que transforma un mensaje hacia atrás posterior en uno anterior. Un ecuación similar se cumple para el nuevo mensaje hacia atrás *después* de que llegue la siguiente observación:

$$\mathbf{b}_{t-d+2:t+1} = \left(\prod_{i=1t-d+2}^{t+1} \mathbf{TO}_i \right) \mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1}\mathbf{1} \tag{15.13}$$

Examinando la expresión producto en la Ecuación (15.12) y (15.13), vemos que tienen una relación sencilla: para obtener el segundo producto, «dividir» el primer producto por el primer elemento $\mathbf{TO}_{t-d+1}$, y multiplicar por el nuevo último elemento $\mathbf{TO}_{t+1}$. En lenguaje matricial, entonces, hay una relación sencilla entre las matrices **B**, la vieja y la nueva:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1}^{-1}\mathbf{B}_{t-d+1:t}\mathbf{TO}_{t+1} \tag{15.14}$$

Esta ecuación proporciona una actualización incremental para la matriz **B**, la cual a su vez (a través de la Ecuación (15.13)) nos permite calcular el nuevo mensaje hacia atrás $\mathbf{b}_{t\text{-}d+2:t+1}$. El algoritmo completo, que requiere almacenar y actualizar **f** y **B**, se muestra en la Figura 15.6.

```
función SUAVIZADO-DE-LAPSO-FIJO (e_t,mom,d) devuelve una distribución sobre X_{t-d}.
  entradas: e_t, la evidencia actual para el paso de tiempo t
            mom, un modelo oculto de Markov con una matriz de transición T de dimensión S × S
            d, la longitud del lapso para el suavizado
  estáticas: t, el tiempo actual, inicialmente 1
            f, una distribución de probabilidad, el mensaje hacia delante P(X_t|e_{1:t}), inicialmente PRIORI[mom]
            B, la matriz de transformación hacia atrás de d-pasos, inicialmente la matriz identidad
            e_{t-d:t}, lista de evidencias desde t − d hasta t, inicialmente vacía
  variables locales: O_{t-d}, O_t, matrices diagonales que contiene la información del modelo sensor

  añadir e_t al final de e_{t-d:t}
  O_t ← matriz diagonal conteniendo P(e_t|X_t)
  si t > d entonces
    f ← HACIA-DELANTE (f, e_t)
    borrar e_{t-d-1} del comienzo de e_{t-d:t}
    O_{t-d} ← matriz diagonal conteniendo P(e_{t-d}|X_{t-d})
    B ← O_{t-d}^{-1} T^{-1} B T O_t
    en otro caso B ← B T O_t
    t ← t+1
    si t > d entonces devolver NORMALIZAR(f × B1) en otro caso devolver nulo (null)
```

Figura 15.6 Un algoritmo para suavizado con un lapso de tiempo fijado de d pasos, implementado como un algoritmo *online* que produce la nueva estimación suavizada dada la observación para un nuevo paso de tiempo.

15.4 Filtros de Kalman

Imagine que mira un pájaro pequeño volando a través del follaje de una jungla densa al anochecer: vislumbra destellos breves e intermitentes del movimiento; intenta con dificultad adivinar dónde está el pájaro y dónde aparecerá la próxima vez para que no lo pierda. O imagine que es un operador de radar en la Segunda Guerra Mundial que mira detenidamente en una apenas visible señal luminosa sin rumbo fijo que aparece cada 10 segundos en la pantalla. O, retrocediendo más lejos aún, imagine que es Kepler intentando reconstruir los movimientos de los planetas de un conjunto de observaciones angulares altamente inexactas tomadas en intervalos medidos de forma irregular e imprecisa. En todos los casos, está intentando estimar el estado (posición y velocidad, por ejemplo) de un sistema físico a partir de observaciones ruidosas en el tiempo. El problema puede formularse como inferencia en un modelo probabilista temporal, donde el modelo de transición describe la física del movimiento y el modelo sensor describe el proceso de medición. Esta sección examina las representaciones especiales y los algoritmos de inferencia que se han desarrollado para resolver estos tipos de problemas; el método que abordaremos se llama **filtrado de Kalman**, después de su inventor, Rudolf E. Kalman.

FILTRADO DE KALMAN

Claramente, necesitaremos sendas variables *continuas* para especificar los estados del sistema. Por ejemplo, el vuelo del pájaro puede especificarse por la posición (X, Y, Z) y la velocidad $(\dot{X}, \dot{Y}, \dot{Z})$ en cada punto en el tiempo. También necesitamos densidades condicionales adecuadas para representar los modelos transición y sensor; como en el Capítulo 14, usaremos distribuciones gaussianas lineales. Esto significa que el siguiente estado $\mathbf{X}_{t+1}$ debe ser una función lineal del estado actual $\mathbf{X}_t$, más algún ruido gaussiano, una condición que resulta ser bastante razonable en la práctica. Considere, por ejemplo, la coordenada X del pájaro, ignorando las otras coordenadas por ahora. Sea Δ el intervalo entre las observaciones, y supongamos velocidad constante; entonces la posición actualizada viene dada por

$$X_{t+\Delta} = X_t + \dot{X}\,\Delta$$

Si añadimos ruido gaussiano entonces tenemos un modelo de transición gaussiano:

$$P(X_{t+\Delta} = x_{t+\Delta} | X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t\Delta, \sigma)(x_{t+\Delta})$$

La estructura de la red bayesiana para un sistema con posición $\mathbf{X}_t$ y velocidad se muestra en la Figura 15.7. Nótese que esta es una forma muy concreta de modelo gaussiano lineal; la forma general se describirá posteriormente en esta sección y abarca una amplia serie de aplicaciones además de los ejemplos de movimiento sencillos del primer párrafo. El lector puede desear consultar el Apéndice A para algunas propiedades matemáticas de las distribuciones gaussianas; para nuestros propósitos inmediatos, el más importante es que una distribución **gaussiana multivariante** para d variables está especificada por un media $\boldsymbol{\mu}$ de d-elementos y la matriz de covarianzas $\boldsymbol{\Sigma}$ de dimensión $d \times d$.

Actualización de distribuciones gaussianas

En el Capítulo 14, nos referimos a una propiedad clave de la familia de distribuciones gaussianas lineales: permanecen cerradas bajo las operaciones estándares de redes bayesianas. Aquí, hacemos esta afirmación en el contexto de filtrado en un modelo probabilista temporal. Las propiedades necesarias corresponden al cálculo del filtrado en dos-pasos de la Ecuación (15.3):

Figura 15.7 Estructura de red bayesiana para un sistema dinámico lineal con posición $\mathbf{X}_t$, velocidad $\dot{\mathbf{X}}_t$, y medida de posición $\mathbf{Z}_t$.

1. Si la distribución actual $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ es gaussiana y el modelo de transición $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$ es gaussiana lineal, entonces la distribución predictiva de un-paso dada por

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t})d\mathbf{x}_t \tag{15.15}$$

   es también una distribución gaussiana.
2. Si la distribución predictiva $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ es gaussiana y el modelo sensor $\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})$ es gaussiana lineal, entonces, después de condicionar a la nueva evidencia, la distribución actualizada

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) \tag{15.16}$$

   es también una distribución gaussiana.

Así, el operador HACIA-DELANTE para el filtrado de Kalman toma un mensaje hacia delante gaussiano $\mathbf{f}_{1:t}$, especificada por una media μ_t y matriz de covarianzas Σ_t, y produce un nuevo mensaje hacia delante gaussiano multivariante $\mathbf{f}_{1:t+1}$, especificado por una media μ_{t+1} y matriz de covarianzas Σ_{t+1}. Así, si empezamos con una *a priori* gaussiana $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0) = N(\mu_0, \Sigma_0)$, filtrando con un modelo gaussiano lineal produce una distribución de estados gaussiana para todo los instantes.

Esto parece ser un resultado bonito y elegante, pero ¿por qué es tan importante? La razón es que, excepto para unos pocos casos especiales tales como este, *el filtrado con redes continuas e híbridas (discretas y continuas) genera distribuciones de estado cuya representación crece sin límite en el tiempo*. Esta afirmación no es fácil de probar en general, pero el Ejercicio 15.5 muestra qué ocurre para un ejemplo sencillo.

Un ejemplo unidimensional sencillo

Hemos dicho que el operador HACIA-DELANTE para el filtro de Kalman transforma una gaussiana en una nueva gaussiana. Esto se traduce en calcular una nueva media y matriz de covarianza a partir de las media y matriz de covarianza previas. Deducir la regla de actualización en el caso general (multivariante) requiere bastante álgebra lineal, así que nos limitaremos a un caso univariante sencillo por ahora; después daremos los resultados para el caso general. Incluso para el caso univariante, los cálculos son un tanto tediosos, pero consideramos que vale la pena verlos porque la utilidad de los filtros de Kalman está íntimamente relacionada a las propiedades de las distribuciones gaussianas.

El modelo temporal que consideraremos describe un **camino aleatorio** de una sola variable estado continua X_t con una observación ruidosa Z_t. Un ejemplo podría ser el índice de «confianza del consumidor», que puede ser modelado como una fluctuación mensual aleatoriamente distribuida como una gaussiana y medida por una encuesta a consumidores aleatorios que también introduce ruido de muestreo gaussiano. La distribución *a priori* se supone que es una gaussiana con varianza σ_0^2:

$$P(x_0) = \alpha e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)}$$

(Por simplicidad, en esta sección usaremos el mismo símbolo α para todas las constantes de normalización.)

El modelo de transición simplemente añade una perturbación gaussiana de varianza constante al estado actual:

$$P(x_{t+1}|x_t) = \alpha e^{-\frac{1}{2}\left(\frac{(x_{t+1}-x_t)^2}{\sigma_x^2}\right)}$$

El modelo sensor entonces asume ruido gaussiano con varianza σ_x^2:

$$P(z_t|x_t) = \alpha e^{-\frac{1}{2}\left(\frac{(z_t - x_t)^2}{\sigma_z^2}\right)}$$

Ahora, dada la *a priori* $P(X_0)$, podemos calcular la distribución predictiva de un paso utilizando la Ecuación (15.15):

$$P(x_1) = \int_{-\infty}^{\infty} P(x_1|x_0)P(x_0)dx_0 = \alpha\int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{(x_1-x_0)^2}{\sigma_x^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)} dx_0$$

$$= \alpha\int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{\sigma_0^2(x_1-x_0)^2+\sigma_x^2(x_0-\mu_0)^2}{\sigma_0^2\sigma_x^2}\right)} dx_0$$

COMPLETAR EL CUADRADO

Esta integral parece bastante complicada. La clave para avanzar es notar que el exponente es la suma de dos expresiones que son *cuadráticas* en x_0 y así él mismo es cuadrático en x_0. Un truco sencillo conocido como **completar el cuadrado** permite la reescritura de cualquier cuadrática $ax_0^2 + bx_0 + c$ como la suma de un término al cuadrado $a\left(x_0 - \frac{-b}{2a}\right)^2$ y un término residual $c - \frac{b^2}{4a}$ que es independiente de x_0. El término residual puede sacarse de la integral, lo que nos da

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(c - \frac{b^2}{4a}\right)} \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(a\left(x_0 - \frac{-b}{2a}\right)^2\right)}$$

Ahora la integral es justamente la integral de una gaussiana en todo su rango, la cual es simplemente 1. Así, nos quedamos sólo con el término residual de la expresión cuadrática.

El segundo paso clave es notar que el término residual tiene que ser cuadrático en x_1; de hecho, después de simplificar, obtenemos

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)}$$

Esto es, la distribución predictiva de un paso es una gaussiana con la misma media μ_0 y una varianza igual a la suma de la varianza original y la varianza de transición . Un ejercicio momentáneo de intuición revela que esto es intuitivamente razonable.

Para completar el paso de actualización, necesitamos condicionar a la observación en el primer paso de tiempo, a saber, z_1. De la Ecuación (15.16), esto viene dado por

$$P(z_1|x_1) = \alpha P(z_1|x_1)P(x_1) = \alpha e^{-\frac{1}{2}\left(\frac{(z_1 - x_1)^2}{\sigma_x^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2}\right)}$$

Una vez más, combinamos los exponentes y completamos el cuadrado (Ejercicio 15.6), obteniendo

$$P(x_1|z_1) = \alpha e^{-\frac{1}{2}\left(\frac{\left(x_1 - \frac{(\sigma_0^2 + \sigma_x^2)z_1 + \sigma_z^2\mu_0}{\sigma_0^2 + \sigma_x^2 + \sigma_z^2}\right)^2}{(\sigma_0^2 + \sigma_x^2)\sigma_z^2/(\sigma_0^2 + \sigma_x^2 + \sigma_z^2)}\right)} \tag{15.17}$$

Así, después de un ciclo de actualización, tenemos una nueva distribución gaussiana para la variable estado.

De la fórmula gaussiana de la Ecuación (15.17), vemos que la nueva media y desviación estándar pueden calcularse a partir de la media y desviación estándar viejas como sigue:

$$\begin{aligned}\mu_{t+1} &= \frac{(\sigma_t^2 + \sigma_x^2)z_{t+1} + \sigma_z^2\mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \\ \sigma_{t+1}^2 &= \frac{(\sigma_t^2 + \sigma_x^2)\sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}\end{aligned} \tag{15.18}$$

La Figura 15.8 muestra un ciclo de actualización para valores particulares de los modelos de transición y sensor.

El par precedente de ecuaciones juega exactamente el mismo papel que la ecuación de filtrado general (15.3) o la ecuación de filtrado de MOM (15.10). Debido a la naturaleza de las distribuciones gaussianas, podemos interpretar el cálculo para la nueva media μ_{t+1}como simplemente una *media ponderada* de la nueva observación z_{t+1} y la media vieja μ_t. Si la observación es poco fiable, entonces σ_z^2 es grande y podemos prestar más atención a la media vieja; si la media vieja es poco fiable (σ_z^2 es grande) o el proceso es altamente impredecible (σ_z^2 es grande), entonces prestamos más atención a la observación. Segundo, nótese que la actualización para la varianza σ_{t+1}^2 es *independiente de la observación*. Podemos así calcular por adelantado qué secuencia de valores de varianza se tendrán. Tercero, la secuencia de valores de varianza converge rápidamente a un valor fijo que depende sólo de σ_x^2 y σ_z^2, consecuentemente se simplifica sustancialmente los cálculos de la secuencia. (*Véase* Ejercicio 15.7.)

Figura 15.8 Etapas en el ciclo de actualización del filtro de Kalman para un camino aleatorio con una *a priori* dada por $\mu_0 = 0{,}0$ y $\sigma_0 = 1{,}0$, ruido de transición dado por $\sigma_x = 2{,}0$, ruido del sensor dado por $\sigma_z = 1{,}0$, y una primera observación $z_1 = 2{,}5$ (marcado en el eje x). Nótese cómo la predicción $P(x_1)$ está achatada, comparada con $P(x_0)$, por el ruido de transición. Nótese también que la media de la posterior $P(x_1|z_1)$ está ligeramente a la izquierda de la observación z_1 porque la media es una media ponderada de la predicción y la observación.

El caso general

La deducción anterior ilustra la propiedad clave de las distribuciones gaussianas que hace que funcione el filtro de Kalman: el hecho de que el exponente es una forma cuadrática. Esto es verdad no sólo para el caso univariante: la distribución gaussiana multivariante tiene la forma

$$N(\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}) = \alpha e^{-\frac{1}{2}((\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}))}$$

Multiplicando los términos del exponente es claro que el exponente es también una función cuadrática de las variables aleatorias x_i de $\mathbf{x}$. Como en el caso univariante, la actualización del filtrado preserva la naturaleza gaussiana de la distribución de estados.

Primero definimos el modelo temporal general usado con el filtrado de Kalman. Tanto el modelo de transición como el modelo sensor tienen en cuenta una transformación *lineal* con ruido gaussiano aditivo. Así, tenemos

$$\begin{aligned} P(\mathbf{x}_{t+1}|\mathbf{x}_t) &= N(\mathbf{F}\mathbf{x}_t, \boldsymbol{\Sigma}_x)(\mathbf{x}_{t+1}) \\ P(\mathbf{z}_t|\mathbf{x}_t) &= N(\mathbf{H}\mathbf{x}_t, \boldsymbol{\Sigma}_x)(\mathbf{z}_t), \end{aligned} \tag{15.19}$$

donde $\mathbf{F}$ y $\boldsymbol{\Sigma}_x$ son matrices que describen el modelo de transición lineal y la covarianza del ruido de transición y $\mathbf{H}$ y $\boldsymbol{\Sigma}_z$ son las matrices correspondientes para el modelo sensor. Ahora las ecuaciones de actualización para la media y covarianza, en su totalidad, horriblemente horripilantes, son

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu} + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \boldsymbol{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1})(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^\top + \boldsymbol{\Sigma}_x) \end{aligned} \tag{15.20}$$

MATRIZ DE GANANCIA DE KALMAN

donde $\mathbf{K}_{t+1} = (\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^\top + \boldsymbol{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^\top + \boldsymbol{\Sigma}_x)\mathbf{H}^\top + \boldsymbol{\Sigma}_z)^{-1}$ se llama **matriz de ganancia de Kalman.** Lo crea o no, estas ecuaciones tienen algún sentido intuitivo. Por ejemplo, considere la actualización para la estimación del estado de la media $\boldsymbol{\mu}$. El término $\mathbf{F}\boldsymbol{\mu}_t$ es el estado *pronosticado* en $t + 1$, así $\mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$ es la observación *pronosticada*. Así, el término $\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$ representa el error en la observación pronosticada. Esto se multiplica por $\mathbf{K}_{t+1}$ para corregir el estado pronosticado; así $\mathbf{K}_{t+1}$ es una medida de *cómo de serio tomar la nueva observación* concerniente a la predicción. Como en la Ecuación (15.18), también tenemos la propiedad de que la actulización de la varianza es independiente de las observaciones. La secuencia de valores para $\boldsymbol{\Sigma}_t$ y $\mathbf{K}_t$ puede así calcularse *offline*, y los cálculos sustanciales necesarios durante el proceso *online* son bastante modestos.

Para mostrar estas ecuaciones en funcionamiento, las hemos aplicado al problema de seguir el movimiento de un objeto en el plano X-Y. Las variables de estado son $\mathbf{X} = (X, Y, \dot{X}, \dot{Y})^\top$ así $\mathbf{F}$, $\boldsymbol{\Sigma}_x$, $\mathbf{H}$, y $\boldsymbol{\Sigma}_z$ son matrices 4×4. La Figura 15.9(a) muestra la trayectoria verdadera, una serie de observaciones ruidosas, y la trayectoria estimada por el filtrado de Kalman, junto con las covarianzas indicadas por los contornos de una-desviación-estándar. El proceso de filtrado hace un buen trabajo de rastreo de la pista del movimiento real, y, como era esperable, la varianza rápidamente alcanza un punto fijo.

Podemos también deducir ecuaciones para el *suavizado* al igual que con el filtrado con modelos gaussianos lineales. Los resultados del suavizado se muestran en la Figura 15.9(b). Nótese cómo la varianza en la estimación de la posición se reduce brusca-

Figura 15.9 (a) Resultados del filtrado de Kalman para el movimiento de un objeto en el plano *X-Y*, mostrando la trayectoria real (de izquierda a derecha), una serie de observaciones ruidosas, y la trayectoria estimada por el filtrado de Kalman. Las varianzas en la estimación de la posición se indica por óvalos. (b) Los resultados del suavizado de Kalman para la misma secuencia de observaciones.

mente, excepto al final de la trayectoria (¿por qué?), y que la trayectoria estimada es mucho más suave.

Aplicabilidad del filtrado de Kalman

El filtro de Kalman y sus modificaciones se utilizan en una amplia serie de aplicaciones. La aplicación «clásica» es en seguimiento por radar de aviones y misiles. Aplicaciones relacionadas incluyen seguimiento acústico de submarinos y vehículos terrestres y seguimiento visual de vehículos y gente. En un filón ligeramente más esotérico, los filtros de Kalman se utilizan para reconstruir trayectorias de partículas a partir de fotografías de cámaras-de-burbujas y flujos oceánicos a partir de mediciones de superficie de satélite. El campo de aplicación es mucho más grande que sólo el seguimiento de movimiento: cualquier sistema caracterizado por variables de estados continuos y mediciones ruidosas lo será. Tales sistemas incluyen fábricas de pulpa, plantas químicas, reactores nucleares, ecosistemas de plantación, y economías nacionales.

FILTRO DE KALMAN EXTENDIDO (FKE)

El hecho de que el filtrado de Kalman pueda aplicarse a un sistema no significa que el resultado será válido o útil. Las suposiciones que se consideran (modelos de transición y sensor son gaussianos lineales) son muy fuertes. El **filtro de Kalman extendido (FKE)** intenta superar la no linealidad en los sistemas a modelar. Un sistema es no lineal si el modelo de transición no puede describirse como una multiplicación matricial del vector de estados, como en la Ecuación (15.19). El FKE trabaja modelando el sistema *localmente* como si fuera lineal en $\mathbf{x}_t$ en la región $\mathbf{x}_t = \boldsymbol{\mu}_t$, la media de la distribución de estados en curso. Esto trabaja bien para suavizado, sistemas con buen-comportamiento y permite al

rastreador conservar y actualizar un distribución de estados gaussiana que es una aproximación razonable para la auténtica *a posteriori*.

¿Qué significa para un sistema ser «no-suavizado» o con «pobre comportamiento»? Técnicamente, significa que hay una no-linealidad significativa en la respuesta del sistema dentro de la región que está «cerrada» (de acuerdo a la covarianza Σ_t) para la media actual $\boldsymbol{\mu}$. Para entender esta idea en términos no técnicos, considere el ejemplo de intentar seguir la pista de un pájaro que vuela entre la jungla. Parece que el pájaro va de cabeza a alta velocidad y en línea recta al tronco de un árbol. El filtro de Kalman, ya sea el normal o el extendido, puede sólo hacer una predicción gaussiana de la localización del pájaro, y la media de esta gaussiana estará centrada en el tronco, como se muestra en la Figura 15.10(a). Un modelo razonable para el pájaro, por otro lado, predeciría una acción evasiva a un lado o a otro, como se muestra en la Figura 15.10(b). Tal modelo es altamente no lineal, ya que la decisión del pájaro varía bruscamente dependiendo de su localización exacta relativa al tronco.

Para manejar ejemplos como este, claramente necesitamos un lenguaje más expresivo para la representación del comportamiento del sistema que va a ser modelado. En la comunidad de la teoría de control, en cuyos problemas tales como maniobras evasivas de aviones se plantean los mismos tipos de dificultades, la solución estándar es el **filtro de Kalman de intercambio**. En este enfoque, los filtros de Kalman múltiples se ejecutan en paralelo, cada uno utilizando un modelo diferente del sistema, por ejemplo, uno para el vuelo en línea recta, uno para girar a la izquierda deprisa, y uno para girar a la derecha deprisa. Se utiliza una suma ponderada de predicciones, donde el peso depende de cómo de bien se ajusta cada filtro a los datos actuales. Veremos en la sección siguiente que es simplemente un caso especial del modelo general de redes bayesianas dinámicas, obtenido mediante la suma de un variable de estado discreta «de maniobras»

FILTRO DE KALMAN DE INTERCAMBIO

Figura 15.10 Un pájaro vuela hacia un árbol (visto desde arriba). (a) Un filtro de Kalman predecirá la localización del pájaro usando una gaussiana simple centrada en el obstáculo. (b) Un modelo más realista tiene en cuenta la acción evasiva del pájaro, y predice que volará a un lado o al otro.

para la red que se muestra en la Figura 15.7. Los filtros de Kalman de intercambio se abordan más en el Ejercicio 15.5.

15.5 Redes bayesianas dinámicas

RED BAYESIANA DINÁMICA

Una **red bayesiana dinámica**, o **RBD**, es una red bayesiana que representa un modelo probabilista temporal del tipo descrito en el Apartado 15.1. Ya hemos visto ejemplos de RBDs: la red del paraguas de la Figura 15.2 y la red del filtro de Kalman de la Figura 15.7. En general, cada corte de una RBD puede tener cualquier número de variables de estado $\mathbf{X}_t$ y variables de evidencia $\mathbf{E}_t$. Por sencillez, supondremos que las variables y sus enlaces están duplicadas exactamente de corte en corte y que la RBD representa un proceso de Markov de primer orden, así que cada variable puede tener padres sólo en su propio corte o el corte inmediatamente anterior.

Debería estar claro que cada modelo oculto de Markov puede representarse como una RBD con una sola variable de estado y una sola variable de evidencia. Es también el caso de cada RBD de variables discretas que puede representarse como un MOM; como se explicó en el Apartado 15.3, podemos combinar todas las variables de estado en la RBD en una sola variable de estado cuyos valores son todas las posibles tuplas de valores de las variables de estado individuales. Ahora, si cada MOM es una RBD y cada RBD puede traducirse en un MOM, ¿cuál es la diferencia? La diferencia es que, *en la descomposición del estado de un sistema complejo en sus variables constituyentes, la RBD es capaz de aprovecharse de la poca densidad del modelo probabilista temporal*. Suponga, por ejemplo, que una RBD tiene 20 variables de estado Booleanas, cada una de las cuales tiene tres padres en el corte anterior. Entonces el modelo de transición de la RBD tiene $20 \times 2^3 = 160$ probabilidades, mientras que el correspondiente MOM tiene 2^{20} estados y así 2^{40} probabilidades, o más o menos un trillón, en la matriz de transición. Esto es malo por al menos tres razones: primero, el MOM en sí mismo requiere mucho más espacio; segundo, la enorme matriz de transición hace la inferencia de MOM mucho más costosa; y tercero, el problema del aprendizaje con tan enorme número de parámetros hace que el modelo MOM puro sea inadecuado para grandes problemas. La relación entre las RBDs y los MOMs es más o menos análoga a la relación entre las redes bayesianas ordinarias y distribuciones conjuntas completas expresadas como tablas.

Ya hemos explicado que cada modelo del filtro de Kalman puede representarse en una RBD con variables continuas y distribuciones condicionales gaussianas lineales (Figura 15.7). Debería estar claro de la discusión del final de la sección anterior que *no* toda RBD puede representarse por un modelo de filtro de Kalman. En un filtro de Kalman, la distribución del estado en curso es siempre una sola distribución gaussiana multivariante, esto es, un solo «bulto» en un lugar concreto. Las RBDs, por otro lado, pueden modelar distribuciones arbitrarias. Para muchas aplicaciones del mundo real, esta flexibilidad es esencial. Considere, por ejemplo, el sitio actual de mis llaves. Deberían estar en mi bolsillo, en la mesita de noche, o en el mostrador de la cocina, o colgadas en la puerta principal. Un solo bulto gaussiano que incluya todos estos lugares debería tener que asignar probabilidad significativa a las llaves que están en medio del aire del vestí-

bulo de entrada. Aspectos del mundo real tales como agentes intencionales, obstáculos, y bolsillos introducen «no-linealidad» que requiere combinaciones de variables discretas y continuas para obtener modelos razonables.

Construcción de RBDs

Para construir una RBD, uno debe especificar tres tipos de información: la distribución *a priori* sobe las variables de estado, $\mathbf{P}(\boldsymbol{X}_0)$; el modelo de transición $\mathbf{P}(\boldsymbol{X}_{t+1}|\boldsymbol{X}_t)$; y el modelo sensor $\mathbf{P}(\boldsymbol{E}_t|\boldsymbol{X}_t)$. Para especificar los modelos transición y sensor, uno debe también especificar la topología de las conexiones entre cortes sucesivos y entre las variables de estado y de evidencia. Ya que se supone que los modelos de transición y sensor son estacionarios (el mismo para todo t) es sencillamente más conveniente especificarlos para el primer corte. Por ejemplo, la especificación completa de la RBD para el mundo del paraguas viene dado por la red de tres nodos que se muestra en la Figura 15.1(a). A partir de esta especificación, la RBD completa (semi-infinita) puede construirse, cuando sea necesario, copiando el primer corte.

Ahora consideremos un ejemplo más interesante: la monitorización de un robot con baterías moviéndose en el plano X-Y, como se introdujo en el Apartado 15.1. Primero, necesitamos variables de estado, que incluirán tanto $\mathbf{X}_t = (X_t, Y_t)$ para la posición y $\dot{\mathbf{X}}_t = (\dot{X}_t, \dot{Y}_t)$ para la velocidad. Supondremos algún método de medición de la posición, quizás una cámara fija o un GPS (*Global Positioning System*) produciendo medidas $\mathbf{Z}_t$. La posición en el siguiente paso de tiempo depende de la posición actual y la velocidad, como en el modelo del filtro de Kalman estándar. La velocidad en el siguiente paso depende de la velocidad actual y el estado de la batería. Añadimos $Bateria_t$ para representar el nivel actual de la carga de la batería, que tiene como padres el nivel de batería previo

Figura 15.11 (a) Especificación de la *a priori*, el modelo de transición y el modelo sensor para la RBD del paráguas. Todos los cortes siguientes se supone que son copias del corte 1. (b) Una RBD sencilla para el movimiento de un robot en el plano X-Y.

y la velocidad previa, y añadimos $BMedidor_t$, que mide el nivel de carga de la batería. Esto nos da el modelo básico que se muestra en la Figura 15.11(b).

Vale la pena mirar con más profundidad en la naturaleza del modelo sensor para $BMedidor_t$. Supongamos, por simplicidad, que tanto $Bateria_t$ y $BMedidor_t$ pueden tomar los valores discretos desde 0 hasta 5 (o mejor como el medidor de batería en un computador portátil típico). Si la medición se hace siempre con precisión, entonces la TPC $\mathbf{P}(BMedidor_t|Bateria_t)$ debería tener probabilidades de 1,0 «a lo largo de la diagonal» y probabilidades de 0,0 en los demás casos. En realidad, el ruido siempre entra en las mediciones. Para mediciones continuas, una distribución gaussiana con una varianza pequeña puede incluso utilizarse[4]. Para nuestras variables discretas, podemos aproximar una gaussiana utilizando una distribución en la que la probabilidad de error se desprenda de modo apropiado, a fin de que la probabilidad de un error grande sea muy pequeña. Usaremos el término **modelo de error gaussiano** para abordar tanto las versiones continuas como las discretas.

MODELO DE ERROR GAUSSIANO

Cualquiera con experiencia práctica de robótica, control de procesos computerizados, u otras formas de percepción automática testificará sin esfuerzo el hecho de que pequeñas cantidades de medición de ruido son con frecuencia el menor de los problemas. Los sensores reales *fallan*. Cuando un sensor falla, necesariamente no manda una señal diciendo, «Ah, a propósito, los datos que le envío son una sarta de disparates». En vez de eso, simplemente envía los disparates. El tipo más sencillo de fallo se llama **fallo transitorio**, donde el sensor ocasionalmente decide enviar algo sin sentido. Por ejemplo, el sensor del nivel de batería puede tener la costumbre de enviar un cero cuando alguien tropieza con el robot, incluso si la batería está totalmente cargada.

FALLO TRANSITORIO

Veamos qué sucede cuando tiene lugar un fallo transitorio con un modelo de error gaussiano que no se adecua a tales fallos. Suponga, por ejemplo, que el robot está tranquilamente posado y observa 20 lecturas de batería consecutivas de valor 5. Entonces el medidor de batería tiene una parada temporal y la siguiente lectura es $BMedidor_{21} = 0$. ¿Qué nos hará creer el modelo de error gaussiano simple sobre $BMedidor_{21}$? Atendiendo a la regla de Bayes, la respuesta depende tanto del modelo sensor $\mathbf{P}(BMedidor_{21} = 0|Bateria_{21})$ como del modelo de predicción $\mathbf{P}(Bateria_{21}|BMedidor_{1:20})$. Si la probabilidad de un error grande del sensor es significativamente menos creible que la probabilidad de una transición a $Bateria_{21} = 0$, incluso si éste es muy poco creíble, entonces la distribución *a posteriori* asignará una alta probabilidad a que la batería está vacía. Una segunda lectura de cero en $t = 22$ tomará esta conclusión casi con certidumbre. Si entonces el fallo transitorio desaparece y la lectura devuelve 5 de $t = 23$ hacia delante, la estimación para el nivel de batería rápidamente vuelve a 5, como por arte de magia. Esta transición de sucesos se ilustra en la curva superior de la Figura 15.12(a), que muestra el valor esperado de $Bateria_t$ en el tiempo utilizando un modelo de error gaussiano discreto.

A pesar de la recuperación, hay un instante ($t = 22$) en el que el robot está convencido de que su batería está vacía; es de suponer que, entonces, mandaría una señal de

[4] Estrictamente hablando, una distribución gaussiana es problemática porque asigna probabilidades no nulas a grandes niveles de carga negativa. La **distribución beta** es a veces una elección mejor para una variable con rango restringido.

Figura 15.12 (a) Curva superior: trayectoria del valor esperado de $Bateria_t$ para una secuencia de observaciones con todos los valores a 5 excepto para $t = 21$ y $t = 22$ que tienen valor 0, usando un modelo de error gaussiano simple. Curva inferior: la trayectoria cuando las observaciones permanecen a valor 0 desde $t = 21$ en adelante. (b) El mismo experimento se realiza con el modelo de fallo transitorio. Nótese que el fallo transitorio se procesa bien, pero el fallo persistente tiene como resultado un pesimismo extremo.

llamada de ayuda y se apagaría. Lamentablemente, su modelo sensor demasiado simplista le ha llevado por mal camino. ¿Cómo puede arreglarse esto? Considere un ejemplo familiar de conducción: en una curva cerrada o en una cuesta pronunciada, la luz de advertencia de «tanque de gasolina vacío» se enciende a veces. En vez de buscar un teléfono de emergencia, uno simplemente recuerda que el indicador de gasolina, a veces, da un error muy grande cuando la gasolina se está agitando en el tanque. La moraleja de esta historia es la siguiente: *Lo indicado para que el sistema manipule fallos del sensor correctamente es que el modelo sensor debe incluir la posibilidad de fallo*.

El tipo más sencillo de modelo de fallos para un sensor permite una probabilidad positiva de que el sensor devolverá indiscutiblemente algún valor incorrecto, sin tener en cuenta el estado real del mundo. Por ejemplo, si el medidor de batería devuelve 0, debemos decir que:

$$P(BMedidor_t = 0 | Bateria_t = 5) = 0{,}03,$$

MODELO DE FALLO TRANSITORIO

lo que se supone mucho más grande que la probabilidad asignada por el modelo de error gaussiano simple. Llamamos a esto **modelo de fallo transitorio**. ¿Cómo ayuda cuando no enfrentamos a una lectura de 0? Supuesto que la probabilidad *de predicción* de una batería vacía, de acuerdo a las lecturas realizadas hasta el momento, es mucho menor que 0,03, la mejor explicación de la observación $BMedidor_{21} = 0$ es que el sensor ha fallado temporalmente. Intuitivamente, podemos pensar en la creencia sobre el nivel de la batería como si tuviera una cierta inercia que ayuda a sobreponerse de las irregularidades en la lectura del medidor. La curva superior de la Figura 15.12(b) muestra que el modelo de fallo transitorio puede manejar fallos transitorios sin un cambio desastroso en las creencias.

Y después de las irregularidades temporales ¿qué hacer si un fallo del sensor persiste? Lamentablemente, los fallos de este tipo son igualmente comunes. Si el sensor devuelve 20 lecturas de 5 seguido de 20 lecturas de 0, entonces el modelo sensor de fallo transitorio descrito en el párrafo anterior redundará en el robot gradualmente llegando a creer que su batería está vacía cuando de hecho puede ser que el medidor esté fallando. La curva inferior de la Figura 15.12(b) muestra la «trayectoria» de la creencia para este caso. Para $t = 25$ (cinco lecturas de 0) el robot está convencido de que su batería está vacía. Obviamente, preferiríamos que el robot crea que su medidor de batería está roto (si es que éste es el suceso más creíble).

MODELO DE FALLOS PERSISTENTES

De modo habitual, para manejar fallos persistentes, necesitaremos un **modelo de fallos persistentes** que describa cómo el sensor se comporta bajo condiciones normales y después de fallos. Para hacer esto, necesitamos aumentar el estado oculto del sistema con una variable adicional, digamos *BMRoto*, que describe el estatus del medidor de la batería. La persistencia de fallos debe modelarse por un arco enlazando $BMRoto_0$ con $BMRoto_1$. Este **arco persistente** tiene una TPC que da una pequeña probabilidad de fallo en cualquier paso de tiempo dado, digamos 0,001, pero especifica que el sensor se queda roto una vez que él se rompe. Cuando el sensor está funcionando perfectamente, el modelo sensor para *BMedidor* es idéntico al modelo de fallo transitorio; cuando el sensor está roto, el modelo dice que *BMedidor* es siempre 0, independientemente de la carga existente en la batería.

ARCO PERSISTENTE

El modelo de fallos persistentes para el sensor de la batería se muestra en la Figura 15.13(a). Su ejecución para las dos secuencias de datos (irregularidades temporales y fallos persistentes) se muestra en la Figura 15.13(b). Hay que observar varias cosas en estas curvas. Primero, en el caso de irregularidad temporal, la probabilidad de que el

Figura 15.13 (a) Un fragmento de RBD mostrando la variable del estatus del sensor necesaria para el modelado del fallo persistente del sensor de la batería. (b) Curvas superiores trayectorias del valor esperado de $Batería_t$ para las secuencias de observaciones «fallo transitorio» y «fallo permanente». Curvas inferiores trayectorias probabilistas para *BMRoto*, dadas las dos secuencias de observaciones.

sensor esté roto crece significativamente después de la segunda lectura de 0, pero inmediatamente se queda en 0 una vez que se observa un 5. Segundo, en el caso de fallos persistentes, la probabilidad de que el sensor esté roto crece rápidamente hasta casi 1 y se queda ahí. Finalmente, una vez que se sabe que el sensor está roto, el robot puede sólo suponer que su batería se descarga a un ritmo «normal», como se muestra por el nivel gradualmente decreciente de $E(Bateria_t|\ldots)$.

Hasta ahora, simplemente hemos arañado la superficie del problema de la representación de procesos complejos. La diversidad de modelos de transición es enorme, abarcando tópicos tan dispersos como modelado del sistema endocrino humano y el modelado de conducción de múltiples vehículos en una autovía. El modelado del sensor es también un amplio subcampo en sí mismo, pero incluso fenómenos sutiles, tales como la tendencia del sensor, descalibrados súbitos, y los efectos de condiciones exógenas (tales como el tiempo climatológico) que afectan a las lecturas del sensor, pueden manipularse mediante representaciones explícitas con redes bayesianas dinámicas.

Inferencia exacta en RBDs

Esbozadas algunas ideas sobre la representación de procesos complejos en RBDs, volvemos ahora al problema de la inferencia. De alguna forma este problema ya se ha resuelto: las redes bayesianas dinámicas *son* redes bayesianas, y ya tenemos algoritmos de inferencia en redes bayesianas. Dada una secuencia de observaciones, uno puede construir la representación de la red bayesiana completa duplicando cortes hasta que la red sea lo suficientemente grande como para adecuarse a las observaciones, como se muestra en la Figura 15.14. Esta técnica se conoce como **desenrollado**. (Técnicamente, la RBD es equivalente a una red semi-infinita obtenida por desenrollado continuo a lo largo del tiempo. Los cortes añadidos más allá de la última observación no tienen efecto en las inferencias dentro del período de observaciones y pueden suprimirse.) Una vez que la RBD está desenrollada, uno puede usar cualquiera de los algoritmos de inferencia (eliminación de variables, métodos de árboles de uniones, etc.) descritos en el Capítulo 14.

DESENROLLADO

Figura 15.14 Desenrollando una red bayesiana dinámica: los cortes son duplicados hasta tener en cuenta la secuencia de observaciones (nodos sombreados). Los cortes posteriores no tienen efecto en las inferencias realizadas en el período de observaciones.

Desafortunadamente, una utilización simplona del desenrollado no sería especialmente eficiente. Si queremos realizar filtrado o suavizado con una larga secuencia de observaciones $\mathbf{e}_{1:t}$, la red desenrollada requeriría espacio $O(t)$ y de este modo crecería sin límites conforme se añadieran más observaciones. Más aún, si cada vez que se añade una observación simplemente ejecutamos de nuevo el algoritmo de inferencia, el tiempo de inferencia por actualización también aumentará según $O(t)$.

Mirando atrás en el Apartado 15.2, vemos que el tiempo y espacio constante por actualización de filtrado puede alcanzarse si los cálculos pueden hacerse de forma recursiva. Esencialmente, la actualización de filtrado en la Ecuación (15.3) funciona *sumando* las variables de estado del paso de tiempo previo para obtener la distribución del nuevo paso de tiempo. La sumatoria de variables es exactamente lo que hace el algoritmo de **eliminación de variables** (Figura 14.10), y resulta que ejecutando la eliminación de variables considerando las variables en su orden temporal, se imita exactamente la operación de la actualización de filtrado recursivo de la Ecuación (15.3). El algoritmo modificado guarda como mucho dos cortes en memoria en cada instante: empezando con el corte 0, añadimos el corte 1, entonces sumamos en el corte 0, después añadimos el corte 2, entonces sumamos en el corte 1, y así sucesivamente. De este modo, podemos conseguir espacio y tiempo constante por actualización de filtrado. (El mismo desarrollo puede lograrse haciendo las modificaciones adecuadas para el algoritmo de árboles de uniones.) El Ejercicio 15.10 le pide que verifique este hecho para la red del paraguas.

Y después de las buenas noticias; ahora las malas noticias: resulta que la «constante» de la complejidad en tiempo y espacio por-actualización es, en casi todos los casos, exponencial en el número de variables de estado. Lo que ocurre es que, como se comporta la eliminación de variables, los factores crecen para incluir todas las variables de estado (o, más concretamente, todas aquellas variables de estado que tienen padres en el corte de tiempo anterior). El tamaño del máximo factor es $O(d^{n+1})$ y el coste de la actualización es $O(d^{n+2})$.

Por supuesto, esto es mucho menor que el coste de la actualización de los MOM, que es $O(d^{2n})$, pero es todavía impracticable para números grandes de variables. Este hecho desalentador es un poco duro de aceptar. Lo que significa es que *aun cuando podamos usar RBDs para representar procesos temporales muy complejos con muchas variables poco conectadas, no podemos razonar eficientemente y exactamente en esos procesos*. El modelo de la RBD en sí, que representa la distribución conjunta *a priori* sobre todas las variables, se divide en factores según las TPCs que la constituyen, pero la distribución conjunta *a posteriori* condicionada a una secuencia de observaciones (eso es, el mensaje hacia delante) generalmente no es divisible en factores. Hasta el momento, nadie ha encontrado un modo alternativo para este problema, a pesar del hecho de que muchas áreas importante de ciencias e ingenierías podrían beneficiarse enormemente de su solución. Así, debemos recurrir a métodos aproximados.

Inferencia aproximada en RBDs

En el Capítulo 14 se describieron dos algoritmos aproximados: ponderación de la verosimilitud (Figura 14.14) y Monte Carlo para cadenas de Markov (MCCM, Figura 14.15).

De los dos, el primero es más fácil de adaptar al contexto de RBDs. Veremos, sin embargo, que se requieren distintas reformas sobre el algoritmo estándar de la ponderación de la verosimilitud antes de que se muestre un método práctico.

Recuerde que la ponderación de la verosimilitud trabaja muestreando los nodos de no-evidencia de la red en orden topológico, ponderando cada muestra por la verosimilitud de acuerdo con las variables de evidencia observadas. Como con los algoritmos exactos, podríamos aplicar la ponderación de la verosimilitud directamente a una RBD desenrollada, pero esto adolecería de los mismos problemas en términos de requerimientos crecientes en tiempo y espacio por actualización conforme la secuencia de observaciones crece. El problema es que el algoritmo estándar pasa cada muestra a su vez, por todos los caminos a través de la red. El algoritmo modificado se adapta al patrón general de los algoritmos de filtrado, con el conjunto de N muestras según el mensaje hacia delante. La primera innovación clave, entonces, es *usar las muestras en sí como una representación aproximada de la distribución de estados en curso*. Esto responde el requerimiento de un tiempo «constante» por actualización, aunque la constante depende del número de muestras requeridas para asegurar una aproximación razonable a la distribución posterior auténtica. Hay también necesidad de desenrollar la RBD, ya que necesitamos tener en memoria sólo el corte en curso y el siguiente corte.

En nuestra discusión sobre la ponderación de la verosimilitud del Capítulo 14, apuntamos a que la precisión del algoritmo se resiente si las variables de evidencia están «más abajo» que las variables que se están muestreando, porque en este caso las muestras se generan sin ninguna influencia de la evidencia. Mirando la estructura típica de una RBD (digamos, la RBD del paraguas de la Figura 15.14) vemos que en efecto las variables de estado anteriores en el tiempo serán muestreadas sin beneficiarse de la evidencia posterior en el tiempo. De hecho, mirando más cuidadosamente, vemos que ¡*ninguna* de las variables de estado tiene *ninguna* de las variables de evidencia entre sus ancestros! Así, aunque el peso de cada muestra dependerá de la evidencia, el conjunto existente de muestras generadas será *completamente independiente* de la evidencia. Por ejemplo, incluso si el jefe trae el paraguas cada día, el proceso de muestreo podría aun así alucinar con un sinfín de días de sol. Lo que significa en la práctica es que la fracción de muestras que permanecen razonablemente cercanas a la sucesión existente de sucesos desciende exponencialmente con t, la longitud de la secuencia de observaciones; en otras palabras, para mantener un nivel de precisión dado, necesitamos aumentar el número de muestras exponencialmente con t. Dado que el algoritmo de filtrado que funciona en tiempo real sólo usa un número fijo de muestras, lo que ocurre en la práctica es que el error estalla después de un número pequeño de pasos de actualización.

Claramente, necesitamos una solución mejor. La segunda innovación clave es *concentrar el conjunto de muestras en regiones de alta-probabilidad en el espacio de estados*. Esto puede hacerse tirando las muestras que tienen muy poco peso, de acuerdo a las observaciones, mientras que multiplicamos aquellas que tiene un peso alto. De ese modo, la población de muestras permanecerá razonablemente cercana a la realidad. Si pensamos en las muestras como un medio para el modelado de la distribución posterior, entonces tiene sentido usar más muestras en regiones del espacio de estado donde la posterior es alta.

FILTRADO DE PARTÍCULAS

Una familia de algoritmos llamada **filtrado de partículas** está diseñada para hacer justo eso. El filtrado de partículas funciona como sigue: primero, se crea una población de N muestras muestreando a partir de la distribución *a priori* en el instante 0, $\mathbf{P}(\mathbf{X}_0)$. Entonces el ciclo de actualización se repite en cada paso de tiempo:

- Cada muestra se propaga hacia delante muestreando el valor del estado siguiente $\mathbf{x}_{t+1}$ dado el valor actual $\mathbf{x}_t$ para la muestra, y utilizando el modelo de transición $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$.
- Cada muestra se pondera por la verosimilitud que asigna la nueva evidencia, $P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})$.
- La población es remuestreada para generar una nueva población de N muestras. Cada nueva muestra se selecciona de la población actual; la probabilidad de que una muestra concreta sea seleccionada es proporcional a su peso. Las muestras nuevas son no-ponderadas.

El algoritmo se muestra con detalle en la Figura 15.15, y su funcionamiento para la RBD del paraguas se ilustra en la Figura 15.16.

Podemos demostrar que este algoritmo es consistente (da las probabilidades correctas cuando N tiende al infinito) considerando qué ocurre durante un ciclo de actualización. Supondremos que la población de la muestra empieza con una representación correcta del mensaje hacia delante $\mathbf{f}_{1:t}$ en el instante t. Escribiendo $N(\mathbf{x}_t|\mathbf{e}_{1:t})$ como el número de muestras empleadas en el estado $\mathbf{x}_t$ después de que las observaciones $\mathbf{e}_{1:t}$ se hayan procesado, tenemos que

$$N(\mathbf{x}_t|\mathbf{e}_{1:t})/N = \mathrm{P}(\mathbf{x}_t|\mathbf{e}_{1:t}) \tag{15.21}$$

para N grande. Ahora propagamos cada muestra hacia delante muestreando las variables de estado en $t+1$, dados los valores de la muestra en t. El número de muestras que al-

función FILTRADO-DE-PARTICULAS (**e**, N, *rbd*) **devuelve** un conjunto de muestras para el siguiente paso de tiempo.
  entradas: **e**, la nueva evidencia entrante
    N, el número de muestras que van a mantenerse
    rbd, una RBD con priori $\mathbf{P}(\mathbf{X}_0)$, modelo de transición $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$, y modelo sensor $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$
  estáticas: S, un vector de muestras de tamaño N, inicialmente generado a partir de $\mathbf{P}(\mathbf{X}_0)$
  variables locales: W, un vector de pesos de tamaño N

  desde $i = 1$ hasta N **hacer**
    $S[i] \leftarrow$ muestrear a partir de $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0 = S[i])$
    $W[i] \leftarrow \mathbf{P}(\mathbf{e}|\mathbf{X}_1 = S[i])$
  $S \leftarrow$ MUESTRA-PONDERADA-CON-REEMPLAZAMIENTO (N, S, W)
  devolver S

Figura 15.15 El algoritmo de filtrado de partículas implementado como una operación de actualización recursiva con estados (el conjunto de muestras). Cada uno de los pasos de muestreo involucra el muestreo de las variables del corte pertinente en orden topológico, en gran medida como en MUESTREO-POR-PRIORI. La operación MUESTRA-PONDERADA-CON-REEMPLAZAMIENTO puede implementarse para ejecutarse en un tiempo esperado de $O(N)$.

Figura 15.16 El ciclo de actualización del filtrado de partículas para la RBD del paraguas con $N=10$, mostrando la población de la muestra de cada estado. (a) En el instante t, ocho muestras indican *Lluvia* y dos indican $\neg$*Lluvia*. Cada una se propaga hacia delante muestreando el estado siguiente vía el modelo de transición. En el instante $t+1$, seis muestras indican *Lluvia* y cuatro indican $\neg$*Lluvia*. (b) Se observa $\neg$*Paraguas* en el instante $t+1$. Cada muestra se pesa por su verosimilitud para la observación, como se indica por el tamaño de los círculos. (c) Se genera un nuevo conjunto de 10 muestras usando una selección aleatoria ponderada a partir del conjunto actual, resultando que dos muestras indican *Lluvia* y ocho que indican $\neg$*Lluvia*.

canzan el estado $\mathbf{x}_{t+1}$ desde el estado $\mathbf{x}_t$ es la probabilidad de transición multiplicada por la población de $\mathbf{x}_t$; así, el número total de muestras que alcanzan $\mathbf{x}_{t+1}$ es

$$N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)N(\mathbf{x}_t|\mathbf{e}_{1:t})$$

Ahora pesamos cada muestra por su verosimilitud para la evidencia en $t+1$. Una muestra en el estado $\mathbf{x}_{t+1}$ recibe peso $P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})$. El peso total de las muestras en $\mathbf{x}_{t+1}$ después de observar $\mathbf{e}_{t+1}$ es así

$$W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t})$$

Ahora para el paso de remuestreo. Ya que cada muestra está copiada con probabilidad proporcional a su peso, el número de muestras en el estado $\mathbf{x}_{t+1}$ después del remuestreo es proporcional al peso total en $\mathbf{x}_{t+1}$ antes del remuestreo:

$$\begin{aligned}
N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1})/N &= \alpha W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) \\
&= \alpha P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) \\
&= \alpha P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)N(\mathbf{x}_t|\mathbf{e}_{1:t}) \\
&= \alpha N P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t}) \qquad \text{(por 15.21)} \\
&= \alpha' P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t}) \\
&= P(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) \qquad \text{(por 15.3)}
\end{aligned}$$

Así la población de la muestra después de un ciclo de actualización representa correctamente el mensaje hacia delante en el instante $t+1$.

Así, el filtrado de partículas es *consistente*, pero ¿es *eficiente*? En la práctica parece que la respuesta es sí: el filtrado de partículas parece asegurar una buena aproximación a la auténtica *a posteriori* utilizando un número constante de muestras. No hay, hasta el momento, garantías teóricas; el filtrado de partículas es en la actualidad un área de

estudio intensivo. Se han propuesto muchas variaciones y mejoras, y el conjunto de aplicaciones está creciendo rápidamente. Ya que es un algoritmo de muestreo, el filtrado de partículas puede utilizarse fácilmente con RBD híbridas y continuas, permitiéndole ser aplicado en áreas como seguimiento de patrones de movimiento complejo en vídeo (Isard y Blake, 1996) y predicción en el mercado de acciones (de Freitas *et al.*, 2000).

15.6 Reconocimiento del habla

RECONOCIMIENTO DEL HABLA

En esta sección veremos una de las más importantes aplicaciones de los modelos probabilistas temporales: **reconocimiento del habla**. La tarea es identificar una secuencia de palabras pronunciadas por un orador, dada la señal acústica. El habla es la modalidad dominante para la comunicación entre los humanos, y el reconocimiento del habla por máquinas de una forma fiable sería inmensamente útil. Todavía más útil sería el **entendimiento del habla** (la identificación del *significado* de la expresión). Para esto, debemos esperar hasta el Capítulo 22.

El habla proporciona nuestro primer contacto con el mundo al natural, sin depurar, de datos de un sensor real. Estos datos son *ruidosos*, muy literalmente: pueden ser ruido de fondo así como resultados artificiales introducidos por el proceso de digitalización; hay variación en el modo en que las palabras se pronuncian, incluso por el mismo orador; diferentes palabras pueden sonar lo mismo; y así sucesivamente. Por estas razones, el reconocimiento del habla se ve como un problema de inferencia probabilista.

En un nivel más general, podemos definir el problema de inferencia probabilista como sigue: sea *Palabras* una variable aleatoria con valores en todas las posibles secuencias de palabras que pueden ser pronunciadas, y sea *señal* la secuencia de la señal acústica observada. Entonces la interpretación más creíble de la expresión es el valor de *Palabras* que maximiza $P(palabras|señal)$. Como es a menudo el caso, la aplicación de la regla de Bayes es útil:

$$P(palabras|señal) = \alpha P(señal|palabras)P(palabra)$$

MODELO ACÚSTICO

HOMÓFONAS

$P(señal|palabras)$ es el **modelo acústico**. Describe los sonidos de las palabras[5], que «ceiling» comienza con una «c» suave y suena lo mismo que «sealing»[6]. (Las palabras que suenan igual se llaman **homófonas**.) $P(palabras)$ se conoce como **modelo del lenguaje**. Especifica la probabilidad *a priori* de cada de cada expresión, por ejemplo que «high ceiling» es mucho más probable que la secuencia de palabras «high sealing»[7].

Los modelos del lenguaje utilizados en los sistemas de reconocimiento del habla son usualmente muy sencillos. El **modelo bigram** que describimos posteriormente en esta sección proporciona la probabilidad de que cada palabra siga a cada una de las otras pala-

[5] En esta sección se recurren a muchos ejemplos de la lengua inglesa. Con objeto de respetar al máximo el trabajo original de los autores **no** se han traducido las palabras que se utilizan como ejemplo, si bien se indicará entre corchetes su significado con la notación [es=] a pie de página y siempre y cuando no se altere en exceso el documento original.

[6] «ceiling» [*es = techo*]. «sealing» [*es = impermeabilización*]

[7] «high ceiling» [*es = techo elevado*]. «high sealing» [*es = fuerte impermeabilización*]

FONOLOGÍA

FONEMAS

bras. El modelo acústico es mucho más complejo. Es un descubrimiento importante del campo de la **fonología** (el estudio de cómo suena el lenguaje), a saber, que todos los lenguajes humanos usan un repertorio limitado de entre 40 y 50 sonidos, llamados **fonemas**. Hablando a grandes rasgos, un fonema es el sonido que corresponde a una sola vocal o consonante, pero hay algunas complicaciones: las combinaciones de letras, tales como «th» y «ng» producen fonemas simples, y algunas letras producen fonemas diferentes en contextos diferentes (por ejemplo, la «a» en *rat* y *rate*[8]). En la Figura 15.17 se listan los fonemas usados en inglés con un ejemplo de cada uno. Un **fonema** es la unidad más pequeña de sonido que tiene un significado distinto para el orador de un lenguaje particular. Por ejemplo, en inglés el fonema «t» en «stick» es el mismo fonema que el fonema «t» de «tick»[9], pero en tailandés se distinguen como dos fonemas separados.

PRONUNCIACIÓN

La existencia de fonemas hace posible dividir el modelo acústico en dos partes. La primera parte trata con la **pronunciación** y especifica, para cada palabra, una distribución de probabilidad sobre posibles secuencias de fonemas. Por ejemplo, «ceiling» se pronuncia [s iy l ih ng], o a veces [s iy l ix ng], o en ocasiones incluso [s iy 1 en]. Los fonemas no son observables directamente, así, a grandes rasgos, el habla está representada como un modelo oculto de Markov cuya variable de estado X_t especifica qué fonema se está pronunciando en el instante t.

Vocales		Consonantes B-N		Consonantes P-Z	
Fonema	Ejemplo	Fonema	Ejemplo	Fonema	Ejemplo
[iy]	b**ea**t	[b]	**b**et	[p]	**p**et
[ih]	b**i**t	[ch]	**Ch**et	[r]	**r**at
[eh]	b**e**t	[d]	**d**ebt	[s]	**s**et
[æ]	b**a**t	[f]	**f**at	[sh]	**sh**oe
[ah]	b**u**t	[g]	**g**et	[t]	**t**en
[ao]	b**ough**t	[hh]	**h**at	[th]	**th**ick
[ow]	b**oa**t	[hv]	**h**igh	[dh]	**th**at
[uh]	b**oo**k	[jh]	**j**et	[dx]	bu**tt**er
[ey]	b**ai**t	[k]	**k**ick	[v]	**v**et
[er]	B**er**t	[l]	**l**et	[w]	**w**et
[ay]	b**uy**	[el]	bott**le**	[wh]	**wh**ich
[oy]	b**oy**	[m]	**m**et	[y]	**y**et
[axr]	din**er**	[em]	bott**om**	[z]	**z**oo
[aw]	d**ow**n	[n]	**n**et	[zh]	mea**s**ure
[ax]	**a**bout	[en]	butt**on**		
[ix]	ros**e**s	[ng]	sin**g**		
[aa]	c**o**t	[eng]	washin**g**	[-]	*silence*

Figura 15.17 El alfabeto fonético DARPA, o ARPAbet, enumerando todos los fonemas utilizados en inglés americano. Hay varias notaciones alternativas, incluyendo un Alfabeto Fonético Internacional (IPA, son las iniciales inglesas), que contiene los fonemas de todas las lenguas conocidas.

[8] «rat» [*es=rata*]. «rate» [*es=índice*]
[9] «stick» [*es=palo*]. «tick» [*es=instante*]

La segunda parte del modelo acústico trata con el modo en que los fonemas se materializan como señales acústicas: esto es, la variable evidencia E_t para el modelo oculto de Markov proporciona las características observadas de la señal acústica en el instante t, y el modelo acústico específico $P(E_t|X_t)$, donde X_t es el fonema actual. El modelo puede tener en cuenta variaciones en el tono, velocidad y volumen, y basarse en técnicas del **procesamiento de la señal** para facilitar descripciones de la señal que sean razonablemente robustas frente a estos tipos de variaciones.

PROCESAMIENTO DE LA SEÑAL

El resto de la sección describe los modelos y algoritmos de abajo arriba, comenzando con señales acústicas y fonemas, las palabras individuales, y finalmente afirmaciones enteras. Concluimos con una descripción de cómo se entrenan todos estos modelos y cómo de bien funcionan los sistemas resultantes.

Sonidos del habla

Las ondas de sonido son cambios periódicos de presión que se propagan a través del aire. Los sonidos pueden medirse con un micrófono cuyo diafragma se desplaza por cambios de presión y genera un flujo diverso continuamente. Un convertidor análogo-a-digital mide el tamaño del flujo, que corresponde a una amplitud de la onda del sonido, en intervalos discretos determinado por la **velocidad de muestreo**. Para el habla, una velocidad de muestreo entre ocho y 16 kHz (es decir, de ocho a 16.000 veces por segundo) es lo normal. (La música de alta calidad muestrea a una velocidad de 44 kHz o más.) La precisión de cada medición está determinada por el **factor de cuantización**; los reconocedores del habla normalmente se mantienen entre ocho y 12 bits. Eso significa que un sistema de características mínimas, muestreando a ocho kHz con ocho-bit de cuantización, requeriría más o menos medio megabyte por minuto de habla. Sería impracticable construir y manipular la distribución $P(señal|fonema)$ con tanta información de señal; así, necesitamos desarrollar descripciones más concisas de la señal acústica.

VELOCIDAD DE MUESTREO

FACTOR DE CUANTIZACIÓN

Primero, observamos que aunque la frecuencia del sonido en el habla puede ser varios kHz, los *cambios* en el contexto de la señal ocurren mucho menos a menudo, quizás en no más de 100 Hz. Así, los sistemas de habla resumen las propiedades de la señal sobre intervalos amplios llamados **marcos**. Una longitud de marco de unos 10 msegs (es decir, 80 muestras a ocho kHz) es lo suficientemente corta como para asegurar que unos pocos fonemas de corta duración se desvanecerán en el proceso de reducción. Dentro de cada marco, representamos lo que está ocurriendo con un vector de **características**. Por ejemplo, podemos querer caracterizar la cantidad de energía en cada uno de los distintos rangos de frecuencias. Otras características importantes incluyen la energía global en un marco y la diferencia con respecto al marco anterior. Seleccionar características a partir de una señal del habla es como escuchar a una orquesta y decir «aquí la corneta francesa está tocando alto y los violines están tocando con suavidad». La Figura 15.18 muestra la secuencia de transformaciones a partir de un sonido al natural hasta una secuencia de marcos. Nótese que los marcos se solapan; esto nos previene de la pérdida de información si ocurriese un evento acústico importante y fuera a caer justo en la frontera de un marco.

MARCOS

CARACTERÍSTICAS

En nuestro ejemplo, hemos mostrado marcos con sólo tres características. Los sistemas reales tienen 10 o incluso cientos de características. Si hay n características y cada una tiene, digamos, 256 valores posibles, entonces el marco se describe por un punto en

Figura 15.18 La traducción de la señal acústica en una secuencia de marcos; cada marco se describe por los valores de tres características acústicas.

el espacio n-dimensional y hay 256^n posibles marcos. Para $n > 2$ sería impracticable representar la distribución $P(\textit{características}|\textit{fonema})$ como una tabla explícita, así que necesitamos más compresión. Hay dos posibles enfoques:

CUANTIZACIÓN DEL VECTOR

- El método de **cuantización del vector**, o CV, divide el espacio n-dimensional en, digamos, 256 regiones etiquetadas desde C1 hasta C256. Cada marco puede entonces representarse con una simple etiqueta en vez de con un vector de n números. Así, la distribución en forma de tabla $P(CV|\textit{fonema})$ tiene 256 probabilidades determinadas por cada fonema. La cuantización del vector no es el más popular en sistemas a gran escala.
- En vez de discretizar el espacio de características, podemos usar una distribución continua parametrizada para describir $P(\textit{características}|\textit{fonema})$. Por ejemplo, podríamos usar una distribución gaussiana con media y matriz de covarianza diferentes para cada uno de los fonemas. Esto trabaja bien si las realizaciones acústicas de cada fonema están agrupadas en una sola región del espacio de características. En la práctica, los sonidos pueden extenderse por varias regiones, y debe utilizarse una **mixtura de gaussianas**. Una mixtura es una suma ponderada de k distribuciones individuales, con lo que $P(\textit{características}|\textit{fonema})$ tiene k pesos, k vectores medias de tamaño n, y k matrices de covarianza de tamaño n^2, esto es, $O(kn^2)$ parámetros por cada fonema.

Por supuesto se pierde alguna información al ir de la señal del habla completa a una etiqueta de la CV o un conjunto de parámetros de una mixtura. El arte del procesamiento de la señal se apoya en la elección de características y regiones (o gaussianas) de forma que la pérdida de información *útil* se minimice. Dado un sonido del habla, éste puede pronunciarse de muchos modos: estridente o suave, rápido o despacio, con tono alto o bajo, con un fondo silencioso o ruidoso, y por cualquiera de los millones de hablantes diferentes, cada uno con un acento diferente y tracto vocal. El procesamiento de la señal confía en eliminar las variaciones manteniendo las partes comunes que definen el sonido[10].

[10] El problema complementario, **identificación del hablante**, elimina las partes comunes y mantiene las diferencias individuales, y entonces intenta hacer coincidir las diferencias con modelos de los hablantes individuales.

Hay dos mejoras más que necesitamos hacer para el modelo sencillo que hemos descrito hasta el momento. El primero trata con la estructura temporal de los fonemas. En el habla normal, la mayoría de los fonemas tienen una duración de 50-100 milisegundos, o 5-10 marcos. El modelo probabilista $P(\textit{características}|\textit{fonemas})$ es el mismo para todos estos marcos, dado que la mayoría de los fonemas tienen una buena estructura interna. Por ejemplo, [t] es una de las distintas **consonantes de parada**, en la que el flujo de aire se corta por un corto período antes de una liberación brusca. Examinando la señal acústica, encontramos que [t] comienza con un silencio, una pequeña explosión en el medio, y (usualmente) un siseo al final. La estructura interna de los fonemas puede captarse por un modelo de **fonemas de tres-estados**; cada fonema tiene los estados Principio, Medio y Final, y cada estado tiene su propia distribución sobre las características.

CONSONANTES DE PARADA

FONEMAS DE TRES-ESTADOS

La segunda mejora se refiere al contexto en el que el fonema es pronunciado. El sonido de un fonema dado puede cambiarse por los fonemas circundantes[11]. Recuerde que los sonidos del habla se producen por el movimiento de los labios, la lengua y mandíbula y forzando el aire a través del tracto vocal. Para coordinar estos movimientos complejos como a razón de cinco o más fonemas por segundo, el cerebro inicia acciones para un segundo fonema antes de que el primero se haya completado y, consecuentemente, la alteración de uno o ambos fonemas. Por ejemplo, en la pronunciación de «sweet» los labios se redondean durante al pronunciación de [s] con antelación al siguiente [w]. Estos **efectos de coarticulación** se capturan parcialmente por el modelo **trifonema**, en el que al modelo acústico para cada fonema se le permite depender del fonema precedente y siguiente. Así, la [w] de «sweet» se escribe [w(s,iy)], es decir, [w] con contexto-izquierdo [s] y contexto-derecho [iy].

EFECTOS DE COARTICULACIÓN

TRIFONEMA

El efecto combinado de los tres-estados en los modelos trifonema hace incrementar el número de posibles estados del proceso temporal desde n fonemas en el alfabeto fonético original ($n \approx 50$ para el ARPAbet) hasta $3n^3$. La experiencia demuestra que la precisión se mejora en vez de compensar el costo extra en términos de inferencia y aprendizaje.

Palabras

Podemos pensar en cada palabra como la especificación de una distribución de probabilidad distinta $P(X_{1:t}|\textit{palabra})$, donde X_i determina el estado del fonema en el i-ésimo marco. Típicamente, dividimos esta distribución en dos partes. El **modelo de pronunciación** que proporciona una distribución sobre la secuencia de fonemas (ignorando tiempos métricos y marcos), mientras que el **modelo del fonema** describe cómo un fonema se pasa a una secuencia de marcos.

Considere la palabra «tomato»[12]. De acuerdo con Gershwin (1937), usted dice [t ow m ey t ow] y yo digo [t ow m aa t ow]. En la parte superior de la Figura 15.19 se muestra un modelo de transición que prevé esta variación. Hay sólo dos posibles caminos a través del modelo, uno corresponde a la secuencia de fonemas [t ow m ey t ow] y la otra

[11] En este sentido, el «modelo del fonema» del habla debería pensarse más como una aproximación útil que como una ley inmutable.
[12] «tomato» [*es* = *tomate*].

Figura 15.19 Dos modelos de pronunciación de la palabra «tomato». Cada modelo se muestra como un diagrama de transición donde los estados son círculos y las flechas muestran las transiciones permitidas con sus probabilidades asociadas. (a) Un modelo que tiene en cuenta diferencias dialécticas. Los números 0,5 son estimaciones basadas en las pronunciaciones preferidas de los dos autores. (b) Un modelo con un efecto de coarticulación en la primera vocal, que permite o el fonema [ow] o el fonema [ah].

a [t ow m aa t ow]. La probabilidad de un camino es el producto de las probabilidades de los arcos que construyen el camino:

$$P([towmeytow]|\text{«tomato»}) = P([towmaatow]|\text{«tomato»}) = 0{,}5$$

El segundo origen de la variación fonética es la **coarticulación**. Por ejemplo, el fonema [t] se produce con la lengua en lo alto de la boca, mientras que [ow] tiene la lengua cerca de la parte baja. Cuando hablamos rápidamente, la lengua a menudo va a una posición intermedia, y obtenemos [t ah] en vez de [t ow]. La parte inferior de la Figura 15.19 proporciona un modelo de pronunciación más complicado para «tomato» que tiene en cuenta estos efectos de coarticulación. En este modelo, hay cuatro caminos distintos, y tenemos

$$P([towmeytow]|\text{«tomato»}) = P([towmaatow]|\text{«tomato»}) = 0{,}1$$
$$P([tahmeytow]|\text{«tomato»}) = P([tahmaatow]|\text{«tomato»}) = 0{,}4$$

Se pueden construir modelos similares para cada palabra que queramos ser capaces de reconocer.

El modelo para un fonema de tres-estados se muestra como un diagrama de transición de estados en la Figura 15.20. El modelo es para un fonema particular, [m], pero todos los fonemas tendrán modelos con topología similar. Para cada estado del fonema, mostramos el modelo acústico asociado, suponiendo que la señal está representada por un etiqueta de CV. Por ejemplo, el modelo afirma que $P(E_t = C_1|X_t = [\text{m}]_{\text{Principio}}) = 0{,}5$. Nótese los auto-

Figura 15.20 Un MOM para el fonema [m] de tres-estados. Cada estado tiene varias posibles salidas, cada una con su propia probabilidad. Las etiquetas desde C_1 hasta C_7 para la CV son arbitrarias.

ciclos de la figura; por ejemplo el estado $[\mathrm{m}]_{\mathrm{Medio}}$ persiste con probabilidad 0.9, lo que significa que el estado $[\mathrm{m}]_{\mathrm{Medio}}$ tiene una duración esperada de 10 marcos. En el modelo tenemos que la duración de cada fonema es independiente de la duración de los otros fonemas; un modelo más sofisticado podría distinguir entre hablar rápido y despacio.

Podemos construir modelos similares para cada fonema, posiblemente dependiendo del contexto trifonema. Cada modelo de palabra, cuando se combinan con los modelos de fonemas, da una especificación completa de un MOM. El modelo especifica las probabilidades de transición entre los estados de los fonemas de marco en marco, así como las probabilidades de las características acústicas para cada estado del fonema.

PALABRAS AISLADAS

Si queremos reconocer **palabras aisladas** (esto es, palabras habladas sin ningún contexto inmediato y con límites claros) entonces necesitamos encontrar la palabra que maximiza

$$P(palabra)|e_{1:t}) = \alpha P(e_{1:t}|palabra)P(palabra)$$

La probabilidad *a priori* $P(palabra)$ puede obtenerse a partir de los datos de texto en curso. $P(e_{1:t}|palabra)$ es la verosimilitud de la secuencia de características acústicas de acuerdo al modelo de palabra. El Apartado 15.2 trató el cálculo de tales verosimilitudes; en concreto, la Ecuación (15.5) proporciona un sencillo cálculo recursivo cuyo coste es lineal en t y en el número de estados de la cadena de Markov. Para encontrar la palabra más creíble, podemos desarrollar este cálculo para cada posible modelo de palabra, multiplicando por la *a priori*, y seleccionar la mejor palabra respectivamente.

Oraciones

DISCURSO CONTINUO

Para tener una conversación con un humano, una máquina necesita ser capaz de reconocer un **discurso continuo** en vez de palabras aisladas. Uno puede imaginar que el dis-

curso continuo no es más que una secuencia de palabras, y para cada una de las cuales podemos aplicar el algoritmo de la sección anterior. Este enfoque falla por dos razones. Primero, ya hemos visto (Apartado 15.2) que la secuencia de las palabras más creíbles no es lo mismo que la secuencia de palabras más creíble. Por ejemplo, en la película *Coge el Dinero y Corre*, un cajero de banco interpreta la nota de un atraco que está descuidadamente escrita por Woody Allen en la que dice «I have a gub». Un modelo bueno de lenguaje sugeriría «I have a gun»[13] como secuencia mucho más creible, incluso aunque la última palabra se parezca más a «gub» que a «gun». La segunda cuestión que debemos afrontar con el discurso continuo es la **segmentación** (el problema de decidir dónde finaliza una palabra y comienza la siguiente). Cualquiera que haya intentado aprender un lenguaje extranjero apreciará este problema: al principio todas las palabras parecen ir juntas. Gradualmente, uno aprende a seleccionar palabras a partir de la confusión de sonidos. En este caso, la primera impresión es correcta; un análisis espectrográfico muestra que en discursos fluidos, las palabras realmente *van* juntas sin silencios entre ellas. Aprendemos a identificar los límites de las palabras a pesar de carecer de silencios.

SEGMENTACIÓN

Comencemos con el modelo del lenguaje, cuyo trabajo en el reconocimiento del habla es especificar la probabilidad de cada una de las secuencias posibles de palabras. Utilizando la notación $w_1,\ldots,w_n$ para indicar una cadena de n palabras y w_i para denotar la i-ésima palabra de la cadena, podemos escribir una expresión para la probabilidad de una cadena con el uso de la regla de la cadena como sigue[14]:

$$P(w_1 \cdots w_n) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1 w_2) \cdots P(w_n|w_1 \cdots w_{n-1}) = \Pi_{i=1}^{n} P(w_i|w_1 \cdots w_{i-1})$$

La mayoría de estos términos son bastante complejos y difíciles de estimar o calcular. Afortunadamente, podemos aproximar esta fórmula con algo más sencillo y que siga capturando una gran parte del modelo del lenguaje. Una versión sencilla, popular y efectiva es el modelo **bigram**. Este modelo aproxima $P(w_i|w_1 \cdots w_{i-1})$ por $P(w_i|w_{i-1})$. En otras palabras, considera la hipótesis de Markov de primer orden para la secuencia de palabras.

BIGRAM

Una gran ventaja del modelo bigram es que el modelo es fácil de entrenar contando el número de veces que cada par de palabras aparecen en un campo representativo de cadenas y utilizando los contadores para estimar las probabilidades. Por ejemplo, si «a» aparece 10.000 veces en el campo de entrenamiento y es seguido por «gun» 37 veces, entonces $\hat{P}(gun_i|a_{i-1}) = 37/10.000$, donde $\hat{P}$ indica la probabilidad estimada. Después de cada entrenamiento uno debería esperar que «I have» y «a gun» tengan altas probabilidades estimadas, mientras que «I has» y «an gun» deberían tener poca probabilidad, La Figura 15.21 muestra algunos contadores de bigram obtenidos de las palabras de este libro.

TRIGRAM

Es posible ir al modelo **trigram** que proporciona valores para $P(w_i|w_{i-1}w_{i-2})$. Éste es un modelo de lenguaje más potente, capaz de discriminar que «ate a banana» es más

[13] «I have a gun» [*es = Tengo una pistola*].

[14] Estrictamente hablando, la probabilidad de una secuencia de palabras depende fuertemente del *contexto* de la expresión; por ejemplo, «I have a gun» es mucho más común en notas que se pasan a un cajero de banco que si está en, digamos, el *Wall Street Journal*. Pocos reconocedores del habla manejan contextos, aparte de los de aprendizaje de un modelo de lenguaje de propósito-especial para una tarea particular.

Palabra	Contador unigram	Palabra anterior							
		of	in	is	on	to	from	model	agent
the	33508	3833	2479	832	944	1365	597	28	24
on	2573	1	0	33	2	1	0	0	6
of	15474	0	0	29	1	0	0	88	7
to	11527	0	4	450	21	4	16	9	82
is	10566	3	6	1	4	2	1	47	127
model	752	8	1	0	1	14	0	6	4
agent	2100	10	3	3	2	3	0	0	36
idea	241	0	0	0	0	0	0	0	0

Figura 15.21 Una tabla parcial de contadores unigram y bigram para las palabras de este libro. «The» es la palabra más común con contador de 33.508 (sobre 513.893 palabras totales). El bigram «of the» es el más común, con 3.833. Algunos contadores son mayores al esperado (por ejemplo cuatro para «on is») ya que los contadores ignoran puntuaciones: una oración debe finalizar con «on» y la siguiente empieza con «is».

creíble que «ate a bandanna»[15]. Para los modelos trigram y hasta una extensión menor para bigram y modelos unigram, hay un problema con los contadores a cero: no necesitaríamos decir que una combinación de palabras es imposible sólo porque no aparecieron en el campo de entrenamiento. El proceso de **suavizado** da un probabilidad no nula a tales combinaciones. Se discute en el Apartado 23.1.

Los modelos bigram y trigram no son tan sofisticados como algunos de los modelos gramaticales que veremos en los Capítulos 22 y 23, pero justifican mejor los efectos locales sensitivos al contexto y se las arreglan para capturar alguna sintaxis local. Por ejemplo, el hecho de que el par de palabras «I has» y «man have» obtengan menos puntuación refleja la convención sujeto-verbo. El problema es que estas relaciones pueden detectarse sólo localmente: «the man have» obtiene poca puntuación, pero «the man with the yellow hat have» no está penalizado.

Ahora consideremos cómo combinar el modelo del lenguaje con los modelos de palabras para que podamos tratar secuencias de palabras correctamente. Asumiremos un modelo de lenguaje bigram por simplicidad. Con tal modelo, podemos combinar todos los modelos de palabras (que están compuestos a su vez de modelos de pronunciación y modelos de fonemas) en un gran modelo MOM. Un estado en MOM de palabra-simple es un marco etiquetado por el fonema en curso y el estado del fonema (por ejemplo $[\mathrm{m}]^{\mathrm{tomato}}_{\mathit{Principio}}$). Si cada palabra tiene una media de p fonema de tres-estados en su modelo de pronunciación, y hay W palabras, entonces el MOM de discurso-continuo tiene $3pW$ estados. Las transiciones pueden ocurrir entre estados de fonemas dentro del fonema dado, entre fonemas de una palabra dada, y entre el estado final de una palabra y el estado inicial de otro. Las transiciones entre palabras suceden según la probabilidad establecida por el modelo bigram.

[15] «ate a banana» [*es=comí un plátano*]. «ate a bandanna» [*es=comí un pañuelo de vivos colores*].

Una vez que hemos construido un MOM combinado, podemos usarlo para analizar la señal del discurso continuo. En particular, el algoritmo de Viterbi contenido en la Ecuación (15.9) puede usarse para encontrar la secuencia de estados más creíble. A partir de esta secuencia de estados, podemos entonces extraer una secuencia de palabras simplemente leyendo las etiquetas de las palabras a partir de los estados. Así, el algoritmo de Viterbi resuelve el problema de segmentación de palabras utilizando programación dinámica para considerar (en consecuencia) todas las secuencias de palabras posibles y fronteras de las palabras simultáneamente.

Nótese que no dijimos «podemos extraer *la* secuencia de palabras *más creible*» . La secuencia de palabras más probable no es necesariamente la que contiene la secuencia de estados más probable. Esto es porque la probabilidad de una secuencia de palabras es la suma de las probabilidades sobre todas las secuencias de estados posibles que sean consistentes con esa secuencia de palabras. La comparación de dos secuencias de palabras, digamos, «a back» y «aback»[16], puede ser ese caso en el que hay 10 alternativas de secuencias de estados para «a back», cada una con probabilidad 0,03, pero sólo una secuencia de estados para «aback», con probabilidad 0,20. Viterbi elige «aback», pero «a back» es, en realidad, más probable.

En la práctica, esta dificultad no es una amenaza de por vida, solamente es lo suficientemente serio como para que se hayan ensayado otros enfoques. El más común es el **descodificador A***, que hace un uso ingenioso de la búsqueda A* (*véase* Capítulo 4) para encontrar la secuencia de palabras más probable. La idea es ver cada secuencia de palabras como un camino a través de un grafo cuyos nodos están etiquetados con palabras. Los sucesores de un nodo son todas las palabras que pueden venir a continuación; así, el grafo para todas las oraciones de longitud n o menos tiene n capas, cada una de anchura como máximo de W, donde W es el número de palabras posibles. Con un modelo bigram, el costo $g(w_1,w_2)$ de un arco entre los nodos etiquetados w_1 hasta w_2 viene dado por $-\log P(w_2|w_1)$; de este modo, el coste del camino total de la secuencia es

DESCODIFICADOR A*

$$Cost(w_1 \cdots w_n) = \sum_{i=1}^{n} -\log P(w_i|w_{i-1}) = -\log \prod_{i=1}^{n} P(w_i|w_{i-1})$$

Con esta definición de costo del camino, la búsqueda del camino más corto es exactamente equivalente a buscar la secuencia de palabras más probable. Para que el proceso sea eficiente, también necesitamos una buena heurística $h(w_i)$ para estimar los costos para completar la secuencia de palabras. Obviamente, esto tiene que hacer algo sobre cuánto de la señal del habla no se ha contemplado todavía por las palabras del camino en curso. Hasta el momento, no se han diseñado heurísticas especialmente interesantes para este problema.

Construcción de un reconocedor del habla

La calidad de un sistema de reconocimiento del habla depende de la calidad de todos sus componentes (el modelo del lenguaje, los modelos de pronunciación de palabras, los modelos de fonemas, y los algoritmos de procesamiento de señal utilizados para obtener características espectrales a partir de la señal acústica). Hemos discutido cómo el mo-

[16] «a back» significa «espalda» o «respaldo». «aback» significa «por sorpresa».

delo del lenguaje puede construirse, y dejamos los detalles del procesamiento de la señal para otros libros de texto. Lo dejamos con los modelos de pronunciación y fonemas. La *estructura* de los modelos de pronunciación (como los modelos del «tomato» de la Figura 15.19) se desarrolla usualmente a mano. Grandes diccionarios de pronunciación están disponibles para el inglés y otros lenguajes, aunque sus precisiones varían enormemente. La estructura de los modelos de fonemas de tres-estados es la misma para todos los fonemas, como se mostró en la Figura 15.20. Eso deja a las probabilidades. ¿Cómo se obtienen éstas, dado que los modelos podrían necesitar cientos de miles o millones de parámetros?

El único método plausible es aprender los modelos a partir de los datos del habla en curso, de lo que ciertamente no falta. La siguiente pregunta es cómo hacer el aprendizaje. Damos la respuesta completa en el Capítulo 20, pero podemos presentar las ideas principales aquí. Considere el modelo de lenguaje biagram; explicamos cómo aprenderlo estudiando las frecuencias de pares de palabras en el texto considerado. ¿Podemos hacer lo mismo para, pongamos, las probabilidades de transición de los fonemas en el modelo de pronunciación? La respuesta es sí, pero sólo si alguien se toma la molestia de anotar cada ocurrencia de cada una de las palabras con la secuencia de fonemas correcta. Esto es una tarea difícil y propensa al error, pero se ha llevado a cabo para algunos conjuntos de datos estándares que contienen varias horas de habla. Si sabemos la secuencia de fonemas, podemos estimar probabilidades de transición para los modelos de pronunciación a partir de las frecuencias de pares de fonemas. Similarmente, si damos el estado del fonema en cada marco (una tarea de etiquetado manual aún más laboriosa) entonces podemos estimar probabilidades de transición para los modelos del fonema. Dado el estado del fonema y las características acústicas de cada marco, podemos también estimar el modelo acústico, o bien directamente a partir de las frecuencias (para modelos de CV) o bien utilizando métodos de ajuste estadístico (para modelos de mixtura de gaussinas; *véase* Capítulo 20).

El coste y lo excepcional de datos etiquetados a mano, y el hecho de que los conjuntos disponibles de datos etiquetados a mano podrían no representar los tipos de oradores y condiciones acústicas que se obtuvieran en un nuevo contexto de reconocimiento, podrían condenar este enfoque al fracaso. *Afortunadamente, el algoritmo de* ***expectación-maximización*** *o EM aprende modelos MOM de transición y sensor sin la necesidad de datos etiquetados*. Estimaciones derivadas de los datos etiquetados a mano pueden utilizarse para inicializar los modelos; después de eso, EM toma el relevo y entrena los modelos en vez de la tarea a mano. La idea es sencilla: dado un MOM y una secuencia de observaciones, podemos usar los algoritmos de suavizado de los Apartados 15.2 y 15.3 para calcular la probabilidad de cada estado en cada paso de tiempo y, por una extensión sencilla, la probabilidad de cada par estado-estado en pasos de tiempo consecutivos. Estas probabilidades pueden verse como *etiquetas inciertas*. A partir de las etiquetas inciertas, podemos estimar nuevas probabilidades de transición y sensor, y se repite el procedimiento EM. El método garantiza el incremento del ajuste entre modelo y datos en cada iteración, y generalmente converge a un conjunto mucho mejor de valores paramétricos que aquellos que se proporcionaron por las estimaciones iniciales, las etiquetadas a mano.

En el estado del arte de los sistema de habla se utilizan conjuntos de datos enormes y recursos computacionales masivos para entrenar los modelos. Para reconocimiento de

palabras aisladas bajo buenas condiciones acústicas (sin ruido de fondo o sin reverberaciones) con un vocabulario de unos pocos cientos de palabras y un solo orador, la precisión puede estar por el 99 por ciento. Para discursos continuos no restringidos con varios oradores, es normal un ajuste entre el 60 por ciento y el 80 por ciento, incluso con buenas condiciones acústicas. Con ruido de fondo y transmisión telefónica, el ajuste se deteriora aún más. Aunque los sistemas aplicados han mejorado continuamente durante décadas, hay todavía cabida para muchas nuevas ideas.

15.7 Resumen

En este capítulo se ha tratado el problema general de la representación y del razonamiento con procesos temporales probabilistas. Los puntos principales son los siguientes:

- El estado cambiante del mundo se trata utilizando un conjunto de variables aleatorias para representar el estado en cada punto en el tiempo.
- Las representaciones pueden diseñarse para satisfacer la **propiedad de Markov**, con el fin de que el futuro sea independiente del pasado dado el presente. Combinada con la hipótesis de que el proceso es **estacionario** (esto es, las dinámicas no cambian en el tiempo), esto simplifica enormemente la representación.
- Un modelo de probabilidad temporal puede pensarse como el formado por un **modelo de transición** que describe la evolución y un **modelo sensor** que describe el proceso de observación.
- Las tareas de inferencia principal en modelos temporales son **filtrado**, **predicción**, **suavizado**, y el cálculo de la **explicación más probable**. Cada una de éstas puede conseguirse utilizando algoritmos recursivos y sencillos con tiempos de ejecución lineales en la longitud de la secuencia.
- Tres familias de modelos temporales se han estudiado con más profundidad: **modelos ocultos de Markov**, **filtros de Kalman** y **redes Bayesiandas dinámicas** (que incluye a los otros dos como casos especiales).
- El **reconocimiento del habla** y el **rastreo** son dos aplicaciones importantes de los modelos de probabilidad temporal.
- A menos que se hagan suposiciones específicas, como en los filtros de Kalman, la inferencia exacta con muchas variables de estado se presenta intratable. En la práctica, el algoritmo de **filtrado de partículas** parece ser un algoritmo aproximado efectivo.

Notas bibliográficas e históricas

Muchas de las ideas básicas para la estimación del estado de sistemas dinámicos vienen del matemático C. F. Gauss (1809), quien formuló un algoritmo determinista de mínimos cuadrados para el problema de la estimación de órbitas a partir de observaciones astronómicas. El matemático ruso A. A. Markov (1913) desarrolló lo que se lla-

mó más tarde la hipótesis de Markov en el análisis de procesos estocásticos; estimó una cadena de Markov de primer orden sobre las letras del texto de *Eugene Onegin*. Un trabajo clasificado como significativo se hizo durante la Segunda Guerra Mundial por Wiener (1942) para procesos de tiempo continuo y por Kolmogorov (1941) para procesos de tiempo discreto. Aunque este trabajo condujo a desarrollos tecnológicos importantes en los 20 años siguientes, el uso de una representación en el dominio de la frecuencia realizaba muchos cálculos bastante engorrosos. El modelado directo en el espacio de estados del proceso estocástico resultó ser más sencillo, como mostraron Swerling (1959) y Kalman (1960). El artículo presentado posteriormente es lo que ahora se conoce como el filtro de Kalman para inferencia hacia delante en sistemas lineales con ruido gaussiano. Resultados importantes sobre suavizado los dedujo Rauch *et al*. (1965), y la impresionante técnica de suavizado denominada de Rauch-Tung-Striebel es todavía un técnica estándar hoy en día. Muchos resultados iniciales fueron recogidos en Gelb (1974). Bar-Shalom y Fortmann (1988) dan un tratamiento más moderno con un sabor Bayesiano, al igual que muchas referencias a la amplia literatura sobre el tema. Chatfield (1989) aborda el enfoque «clásico» del análisis de series temporales.

En muchas aplicaciones del filtrado de Kalman, uno debe tratar no sólo con percepciones y dinámicas inciertas, sino también con *identidad* incierta; esto es, si hay diversos objetos que se están monitorizando, el sistema debe determinar qué observaciones fueron generadas y por qué objetos antes de que pueda actualizar cada una de las estimaciones del estado. Este es el problema de **asociación de datos** (Bar-Shalom y Fortmann, 1988; Bar-Shalom, 1992). Con n observaciones y n caminos (un caso bastante benévolo), hay $n!$ posibles asignaciones de las observaciones a los caminos; un tratamiento probabilista correcto debe tener en cuenta a todas ellas, y puede demostrarse que esto es NP-duro (Cox, 1993; Cox y Hingorani, 1994). Los métodos aproximados de tiempo polinomial basados en MCCM parecen trabajar bien en la práctica (Pasula *et al*., 1999). Es interesante notar que el problema de asociación de datos es un caso particular de la inferencia probabilista en lenguajes de *primer-orden*; a diferencia de la mayoría de los problemas de inferencia probabilista, que son puramente proposicionales, la asociación de datos involucra a *objetos* así como la *relación de identidad*. Está así íntimamente unido a los lenguajes probabilistas de primer-orden que se mencionaron en el Capítulo 14. Trabajos recientes han demostrado que el razonamiento sobre la identidad en general, y la asociación de datos en particular, puede llevarse a cabo dentro del marco probabilista de primer-orden (Pasula y Russell, 2001).

ASOCIACIÓN DE DATOS

El modelo oculto de Markov y los algoritmos asociados para inferencia y aprendizaje, incluyendo el algoritmo hacia delante-atrás, fueron desarrollados por Baum y Petrie (1966). Ideas similares aparecieron también independientemente en la comunidad de los filtrados de Kalman (Rauch *et al*., 1965). El algoritmo hacia delante-atrás fue uno de los precursores principales de la formulación general del algoritmo EM (Dempster *et al*., 1977); *véase* también Capítulo 20. El suavizado en el espacio-constante aparece en Binder *et al*. (1997b), que construye el algoritmo divide y vencerás desarrollado en el Ejercicio 15.3.

Las redes bayesianas dinámicas (RBDs) pueden verse como codificaciones esparcida de un proceso de Markov y se utilizaron por primera vez en IA por Dean y Kana-

zawa (1989b), Nicholson (1992), y Kjaerulff (1992). El último trabajo incluye una extensión genérica al sistema de redes de creencia HUGIN para suplir habilidades necesarias para la generación y compilación de redes bayesianas dinámicas. Las redes bayesianas dinámicas se han hecho populares para el modelado de diversos procesos de movimiento complejo en visión por computador (Huang *et al*., 1994; Intille y Bobick, 1999). El enlace entre MOMs y RBDs, y entre el algoritmo hacia delante-atrás y propagación de redes bayesianas, las hizo explícitas Smyth *et al*. (1997). Una unificación más con los filtros de Kalman (y otros modelos estadísticos) aparece en Roweis y Ghahramani (1999).

El algoritmo de filtrado de partículas descrito en el Apartado 15.5 tiene una historia particularmente interesante. El primer algoritmo de muestreo para el filtrado se desarrolló en la comunidad de la teoría de control por Handschin y Mayne (1969), y la idea de remuestrear, que es el núcleo del filtrado de partículas, apareció en un artículo ruso de control (Zaritskii *et al*., 1975). Fue posteriormente reinventado en estadística como **remuestreo del muestreo-por-importancia secuencial**, o **SIR** en términos ingleses (Rubin, 1988; Liu y Chen, 1998), en teoría de control como filtrado de partículas (Gordon *et al*., 1993; Gordon, 1994), en IA como **supervivencia del mejor ajuste** (Kanazawa *et al*., 1995), y en visión por computador como **condensación** (Isard y Blake, 1996). El artículo de Kanazawa *et al*. (1995) incluye una mejora denominada **inversión de evidencia** por la que el estado en el instante $t + 1$ se muestrea condicionado tanto al estado en el instante t *como a la evidencia en el instante* $t + 1$. Esto permite a la evidencia influir directamente en la generación de la muestra y Doucet (1997) probó que reduce el error de la aproximación.

Otros métodos para filtrado aproximado incluyen el algoritmo **MCCM descompuesto** (Marthi *et al*., 2002) y el método aproximado factorizado de Boyen *et al*. (1999). Estos dos métodos tienen la propiedad importante de que el error de aproximación no diverge en el tiempo. Técnicas variables (*véase* Capítulo 14) también se han desarrollado para modelos temporales. Ghahramani y Jordan (1997) estudian un algoritmo aproximado para el **MCCM factorial**, una RBD en la que dos o más cadenas de Markov evolucionan independientemente y se enlazan por una lista de observaciones compartidas. Jordan *et al*. (1998) cubren otras aplicaciones. Las propiedades de tiempos mezclados se discuten en Pak (2001) y (Luby y Vigoda, 1999).

La prehistoria del reconocimiento del habla comenzó en 1920 con Radio Rex, un juego con forma de perro activado por voz. Rex saltaba como respuesta a frecuencias de sonido próximas a los 500 Hz, lo que corresponde a la vocal [eh] de «¡Rex!». Algo más serio empezó a funcionar después de la Segunda Guerra Mundial. En los Laboratorios AT&T, se construyó un sistema para el reconocimiento de dígitos aislados (Davis *et al*., 1952) mediante coincidencias de patrones sencillos de características acústicas. Las probabilidades de transición de fonemas se utilizaron primero en un sistema construido en University College, Londres, por Fry (1959) y Denes (1959). A principios de 1971, la Defense Advanced Research Projects Agency (DARPA) del Departamento de Defensa de los Estados Unidos financió cuatro proyectos de cinco años, que competían entre sí, para desarrollar sistemas de reconocimiento del hablar de alto desarrollo. El ganador, y el único sistema que alcanzaba el objetivo de tener una precisión del 90 por ciento con un vocabulario de 1.000 palabras, fue el sistema HARPY de la CMU (Lowerre, 1976;

Lowerre y Reddy, 1980)[17]. La versión final de HARPY provenía de un sistema llamado DRAGON construido por el estudiante graduado James Baker (1975) de la CMU; DRAGON fue el primero en usar un MOM para el habla. Casi simultáneamente, Jelinek (1976) de IBM desarrolló otro sistema basado en un MOM. A partir de ese momento en adelante, los métodos probabilistas en general, y los MOMs en particular, dominaron la investigación y desarrollo del reconocimiento del habla. Los años recientes se caracterizan por progresos crecientes, conjuntos de datos y modelos más grandes, y competiciones más rigurosas en tareas sobre el habla más realista. Algunos investigadores han explorado la posibilidad de usar RBDs en vez de MOMs para el lenguaje, con el propósito de utilizar la mayor potencia expresiva de las RBDs para capturar más cosas sobre los estados ocultos complejos del mecanismo del habla (Zweig y Russell, 1998; Richardson *et al.,* 2000).

Existen varios libros de texto buenos sobre el reconocimiento del habla (Rabiner y Juang, 1993; Jelinek, 1997; Gold y Morgan, 2000; Huang *et al.*, 2001). Waibel y Lee (1990) recogen artículos importantes del área, incluyendo algunos tutoriales. La presentación en este capítulo recurre a la visión general de Kay, Gawron y Norvig (1994) y del libro de Jurafsky y Martin (2000). La investigación del reconocimiento del habla se publica en *Computer Speech and Language*, *Speech Communications*, y la IEEE *Transactions on Acoustics, Speech, and Signal Processing* y en los Workshops de DARPA sobre Procesamiento del Lenguaje Natural y el Habla y las conferencias Eurospeech, ICSLP, y ASRU.

EJERCICIOS

15.1 Demuestre que cualquier proceso de Markov de segundo orden puede reescribirse como un proceso de Markov de primer orden con un conjunto aumentado de variables de estado. ¿Puede siempre hacerse *con parsimonia*; es decir, sin incrementar el número de parámetros necesarios para especificar el modelo de transición?

15.2 En este ejercicio examinamos qué le ocurren a las probabilidades del mundo del paraguas en el límite de una larga secuencia de tiempos.

a) Suponga que observamos una secuencia sin fin de días en los que aparece el paraguas. Demuestre que, conforme pasan los días, la probabilidad de llover en el día actual tiene un crecimiento monótono hacia un punto fijo. Calcule este punto fijo.

b) Ahora considere hacer un *pronóstico* más allá en el futuro, dadas sólo las dos primeras observaciones del paraguas. Primero, calcule la probabilidad $P(R_{2+k}|U_1,U_2)$ para $k = 1\ldots20$ y dibuje el resultado. Debería ver que la probabilidad converge hacia un punto fijo. Calcule el valor exacto de este punto fijo.

[17] El que quedó segundo en la competición, HEARSAY-II (Erman *et al.*, 1980), tuvo una gran influencia en otras ramas de la investigación de la IA por el uso de la **arquitectura de pizarra**. Era un sistema experto basado en reglas con un número de **fuentes de conocimiento** modulares, más o menos independientes, que estaban comunicados por una **pizarra** común en la que podían escribir y leer. Los sistemas de pizarra son la base de las modernas arquitecturas de interfaz de usuario.

15.3 Este ejercicio desarrolla una variante de eficiencia de espacio del algoritmo hacia delante-atrás que se describió en la Figura 15.4. Queremos calcular $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ para $k = 1,\ldots,t$. Esto se hará con un enfoque del divide y vencerás.

a) Suponga, por sencillez, que t es impar, y tomemos el punto medio $h = (t + 1)/2$. Demuestre que $\mathrm{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ puede calcularse para $k = 1,\ldots, h$ dado sólo el mensaje hacia delante inicial $\mathbf{f}_{1:0}$, el mensaje hacia atrás $\mathbf{b}_{h+1:t}$, y la evidencia $\mathbf{e}_{1:h}$.

b) Demuestre un resultado similar para la segunda mitad de la secuencia.

c) Dados los resultados de (a) y (b), un algoritmo divide y vencerás recursivo puede construirse: primero ejecutando hacia-delante a lo largo de la secuencia y después hacia-atrás desde el final, almacenando sólo los mensajes necesarios en medio y los finales. Entonces el algoritmo se llama en cada mitad. Escriba el algoritmo al detalle.

d) Calcule la complejidad en tiempo y espacio del algoritmo como una función de t, la longitud de la secuencia. ¿Cómo cambia si dividimos la entrada en más de dos segmentos?

15.4 En el Apartado 15.2, esbozamos un procedimiento deficiente para la búsqueda de la secuencia de estados más creíble, dada una secuencia de observaciones. El procedimiento implica la búsqueda del estado más creíble en cada paso de tiempo, utilizando el suavizado, y devolviendo la secuencia compuesta por estos estados. Demuestre que, para algunos modelos probabilistas temporales y para algunas secuencias de observaciones, este procedimiento retorna una secuencia de estados imposible (es decir, la probabilidad posteriori de la secuencia es cero).

15.5 A menudo, deseamos monitorizar un sistema de estados continuos cuyo comportamiento cambia de forma impredecible entre un conjunto de k «modos» distintos. Por ejemplo, un avión intentando evadir un misil puede ejecutar una serie de maniobras dispares que el misil puede intentar rastrear. Una representación de red bayesiana de tal modelo de **filtro de Kalman de conmutación** se muestra en la Figura 15.22.

a) Suponga que el estado discreto S_t tiene k posibles valores y que la estimación del estado continuo *a priori* $\mathbf{P}(\mathbf{X}_0)$ es una distribución gaussiana multivariable. Demuestre que la predicción $\mathbf{P}(\mathbf{X}_1)$ es una **mixtura de gaussianas**, esto es, una suma ponderada de gaussianas tales que la suma de los pesos es 1.

b) Demuestre que si la estimación de estados continuos actual $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ es una mixtura de m gaussianas, entonces en el caso general la estimación de estados actualizada $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1})$ será una mixtura de km gaussianas.

c) ¿Qué aspectos del proceso temporal representan los pesos en la mixtura gaussiana?

15.6 Complete el paso que falta en la deducción de la Ecuación (15.17), el primer paso de actualización para el filtro de Kalman unidimensional.

15.7 Examinemos el comportamiento de la actualización de la varianza de la Ecuación (15.18).

a) Dibuje el valor de σ_t^2 como una función de t, dados varios valores para σ_x^2 y σ_z^2.

Figura 15.22 Una representación de red bayesiana de un filtro de Kalman de conmutación. La variable de conmutación S_t es una variable de estado discreto cuyo valor determina el modelo de transición para las variables de estado continuas $\mathbf{X}_t$. Para cualquier estado discreto i, el modelo de transición $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, S_t = i)$ es un modelo gaussiano lineal, al igual que en el filtro de Kalman habitual. El modelo de transición para el estado discreto, $\mathbf{P}(S_{t+1}|S_t)$, puede pensarse como una matriz, como en un modelo oculto de Markov.

b) Demuestre que la actualización tiene un punto fijo σ^2 tal que $\sigma_t^2 \rightarrow \sigma^2$ cuando $t \rightarrow \infty$, y calcule el valor de σ^2.

c) Dé una explicación cualitativa para lo que ocurre cuando $\sigma_x^2 \rightarrow 0$ y cuando $\sigma_z^2 \rightarrow 0$.

15.8 Demuestre cómo representar un MOM como un modelo probabilista relacional recursivo, como se sugirió en el Apartado 14.6.

15.9 En este ejercicio, analizamos con más detalle el modelo de fallo persistente para el sensor de la batería de la Figura 15.13(a).

a) La Figura 15.13(b) para en $t = 32$. Describa cualitativamente qué debería ocurrir cuando $t \rightarrow \infty$ si el sensor continúa hasta leer 0.

b) Suponga que la temperatura externa afecta al sensor de la batería de tal modo que los fallos transitorios llegan a ser más creíbles que el incremento de la temperatura. Muestre cómo aumentar la estructura de la RBD de la Figura 15.13(a), y explique cualquier cambio necesario en las TPCs.

c) Dada la nueva estructura de la red, ¿pueden las lecturas de la batería ser utilizadas por el robot para inferir la temperatura actual?

15.10 Considere la aplicación del algoritmo de eliminación de variables para la RBD desenrollada del paraguas para tres cortes, donde la pregunta es $\mathbf{P}(R_3|U_1, U_2, U_3)$. Muestre que la complejidad del algoritmo (el tamaño del factor más grande) es la misma, independientemente de si las variables de la lluvia se eliminan hacia delante o hacia atrás.

15.11 El modelo de «tomato» de la Figura 15.19 permite una coarticulación sobre la primera vocal dando dos posibles fonemas. Un enfoque alternativo es usar el modelo tri-

fonema en el que el fonema [ow(t,m)] incluye automáticamente el cambio en el sonido de la vocal. Trace un modelo trifonema completo para «tomato», incluyendo la variación dialéctica.

15.12 Calcule el camino más probable dentro del MOM de la Figura 15.20 para la secuencia de salida $[C_1, C_2, C_3, C_4, C_5, C_6, C_7]$. Dé también la probabilidad.

16 Toma de decisiones sencillas

Donde se verá cómo debe tomar decisiones un agente para obtener lo que desea, por lo menos, en promedio.

En este capítulo se retoma la teoría de la utilidad introducida en el Capítulo 13 y se muestra cómo combinarla con la teoría de la probabilidad, para construir un agente basado en la teoría de la decisión: aquél capaz de tomar decisiones racionales basándose en lo que cree y desea. Esta clase de agentes puede adoptar decisiones en situaciones en las que un agente lógico no tiene forma de decidir debido a la presencia de incertidumbre y objetivos contradictorios. En efecto, un agente basado en objetivos maneja una dicotomía entre estados buenos (objetivo) y malos (no objetivo), mientras que un agente basado en la teoría de la decisión maneja una medida cuantitativa de la calidad de los estados.

En la Sección 16.1 se presenta el principio básico de la teoría de la decisión: la maximización de la utilidad esperada. En la Sección 16.2 se muestra que el comportamiento de cualquier agente racional puede modelarse con una función de utilidad que debe maximizarse. La Sección 16.3 trata, más en detalle, sobre la naturaleza de las funciones de utilidad y, en particular, su relación con magnitudes cuantitativas individuales, tales como el dinero. En la Sección 16.4 se muestra cómo manejar las funciones de utilidad que dependen de varias magnitudes cuantitativas. En la Sección 16.5 se explica cómo implantar sistemas para la toma de decisiones. En particular, se presenta un formalismo denominado **redes de decisión** (también conocido como **diagramas de influencia**) que amplía la semántica de las redes Bayesianas mediante la incorporación de acciones y utilidades. El resto del capítulo recoge aspectos diversos que surgen al aplicar la teoría de la decisión a los sistemas expertos.

16.1 Combinación de creencias y deseos bajo condiciones de incertidumbre

En la obra *Port-Royal Logic*, escrita en 1662, el filósofo francés Arnauld afirmaba que:

> Para juzgar lo que hay que hacer para alcanzar un bien o para evitar un mal, hace falta considerar no sólo al bien y al mal en sí, sino también la probabilidad de que sucedan o no y, representar geométricamente la proporción que todos estos factores presentan en conjunto.

Si bien los textos actuales que tratan sobre la teoría de la decisión no hacen referencia al bien o al mal sino al término utilidad, el principio es exactamente el mismo que el expresado por la cita referida. Las preferencias de un agente respecto a ciertos estados del mundo se sustentan mediante una **función de utilidad**, que asigna una cantidad numérica para expresar lo deseable que es un estado. Esta medida de la utilidad, combinada con la probabilidad de ocurrencia de las acciones, da como resultado la utilidad esperada para cada acción.

Se utilizará la notación $U(S)$ para denotar la utilidad del estado S, según considere el agente que toma las decisiones. Por ahora, se considerará a los estados como instantáneas completas del mundo, semejantes a las **situaciones** del Capítulo 10. Aunque esta consideración simplificará las explicaciones iniciales, con el tiempo será complicado especificar la utilidad de cada uno de los estados posibles por separado. En la Sección 16.4, se mostrará cómo, bajo determinadas circunstancias, los estados pueden descomponerse con el fin de asignar utilidades.

Los posibles estados resultantes de una acción no determinista A se denotarán por la notación $Resultado_i(A)$, donde el índice i recorre el rango de posibles resultados. Previamente a la realización de la acción A, el agente asigna una probabilidad $P(Resultado_i(A)|Realizar(A), E)$ a cada resultado, donde el factor E engloba la evidencia disponible por el agente sobre el mundo y $Realizar(A)$ se refiere a la proposición consistente en ejecutar la acción A en el estado actual. Puede calcularse entonces la **utilidad esperada** de la acción A dada la evidencia E, denotada por $UE(A|E)$, mediante la siguiente expresión:

UTILIDAD ESPERADA

$$EU(A|E) = \sum_i P(Resultado_i(A)|Realizar(A), E)U(Resultado_i(A)) \qquad (16.1)$$

MÁXIMA UTILIDAD ESPERADA

El principio de la **máxima utilidad esperada** (MUE) establece que un agente racional debe elegir aquella acción que maximice la utilidad esperada del agente. Si se quiere elegir la mejor secuencia de acciones empleando esta ecuación, se tienen que enumerar todas las secuencias de acciones posibles y elegir la mejor. Claramente, esto es intratable en el caso de secuencias extensas. Por lo tanto, este capítulo se centrará en decisiones simples (compuestas por una única acción) y será en el siguiente capítulo donde se traten las técnicas para manejar eficientemente secuencias de acciones.

En cierto sentido, puede considerarse el principio de MUE como definitorio de la IA. Todo lo que tiene que hacer un agente inteligente es calcular varias medidas, maximizar la utilidad de sus acciones e ir hacia delante. Pero esto no quiere decir que el problema de la IA se *resuelva* por la definición.

Aunque el principio de MUE permite establecer la acción correcta a realizar en cualquier problema de decisión, el coste computacional asociado puede ser prohibitivo y, además, en algunos casos es difícil formular completamente el problema a resolver. El conocimiento del estado inicial del mundo requiere capacidades de percepción, aprendizaje, representación del conocimiento e inferencia. El cálculo de la probabilidad $P(Resultado_i(A) \mid Realizar(A), E)$ requiere la disponibilidad de un modelo causal completo del mundo y, tal y como se vió en el Capítulo 14, la inferencia en redes Bayesianas es un problema NP-completo. El cálculo de la utilidad de cada estado, $U(Resultado_i(A))$, a menudo implica la realización de una búsqueda o planificación ya que un agente no sabe lo bueno que es un estado hasta que no conoce cómo llegar a dicho estado. Por lo tanto, la teoría de la decisión no es la panacea que resuelve el problema de la IA. Como contrapartida, proporciona un marco formal dentro del cual tienen cabida todos los componentes de un sistema de IA.

El principio de MUE tiene una clara relación con la idea de la medida de desempeño introducida en el Capítulo 2. La idea básica es muy sencilla. Considérense los diferentes entornos que pueden condicionar a un agente a disponer de una historia de percepciones dada y los diferentes agentes que podrían diseñarse. *Si un agente maximiza una función de utilidad que, de forma adecuada, refleja la medida de desempeño mediante la cual debe juzgarse su comportamiento, entonces se logrará la más elevada puntuación posible en cuanto a desempeño, si promediamos la función de utilidad sobre todos los entornos en los que puede ejecutarse el agente*. Esta es la justificación fundamental del principio de MUE en sí mismo. Mientras esta aseveración puede parecer una tautología, expresa una idea muy importante relativa a la traslación de un criterio de racionalidad externo y global (como es el de la medida de desempeño sobre una base de hechos) a un criterio local e interno que involucra la maximización de una función de utilidad, cuando se aplica a la determinación del estado siguiente.

DECISIONES UNITARIAS

Este capítulo, sólo se referirá a las **decisiones unitarias** o sencillas, mientras que el Capítulo 2 definía las medidas de desempeño con base en un histórico de hechos sobre el entorno, que usualmente implica múltiples decisiones. En el capítulo siguiente, se tratará el caso de **decisiones secuenciales** y se verá cómo pueden conciliarse las dos perspectivas.

16.2 Los fundamentos de la teoría de la utilidad

Intuitivamente, el principio de Máxima Utilidad Esperada (MEU) aparenta ser una forma razonable para tomar decisiones, pero obviamente no es la *única* vía razonable de hacerlo. Después de todo, ¿por qué debería ser tan especial maximizar la utilidad *media*? ¿Por qué no maximizar la suma de los cubos de todas las posibles utilidades o tratar de minimizar las peores pérdidas de utilidad posibles? También, ¿no podría actuar racionalmente un agente sin más que expresar sus preferencias acerca de los diferentes estados, sin necesidad de asignarles un valor numérico? Quizás, un agente racional puede tener una estructura de preferencias demasiado compleja como para ser representada por un único número real asociado a cada estado.

Restricciones sobre preferencias racionales

Las cuestiones anteriores pueden responderse mediante la escritura de ciertas restricciones sobre las preferencias que un agente racional debería tener y demostrando que el principio de MUE puede derivarse a partir de estas restricciones. Se usará la siguiente notación para explicitar las preferencias de un agente:

$A \succ B$ Se prefiere a A sobre B.
$A \sim B$ El agente es indiferente a la hora de escoger entre A y B.
$A \succsim B$ El agente prefiere a A sobre B, o ambos le son indiferentes.

Ahora la cuestión obvia es, ¿qué clase de cosas son A y B? Si las acciones del agente son deterministas, A y B serán los estados concretos y completamente especificados, resultantes de tales acciones. En el caso más general, es decir, el caso no determinista, A y B son lo más parecido a una **lotería**, en el sentido de que representa un proceso aleatorio en el que ocurrirá uno de entre varios resultados posibles: $C_1, \ldots, C_n$, y en los que la ocurrencia de cada resultado vendrá determinado por las probabilidades asociadas a los posibles resultados: $p_1, \ldots, p_n$, respectivamente. Así pues, un experimento aleatorio (lotería) L puede representarse mediante una distribución de probabilidad dada por la expresión:

LOTERÍA

$$L = [p_1, C_1; p_2, C_2; \ldots, p_n, C_n]$$

(Una lotería con un único resultado podría escribirse como A o mediante el suceso seguro $[1, A]$.) En general, cada resultado de una lotería puede ser un estado atómico u otra lotería. El reto principal para la teoría de la utilidad es comprender cómo están relacionadas las preferencias entre loterías complejas con las preferencias entre los estados implícitamente asociados a tales loterías.

Para realizar esto, se imponen una serie de restricciones razonables sobre la relación de preferencia, de la misma forma que en el Capítulo 13 se impusieron restricciones a los grados de creencia para obtener los axiomas de la teoría de la probabilidad. Una restricción razonable sería que la relación de preferencia debería ser **transitiva**, es decir, si $A \succ B$ y $B \succ C$, entonces se esperaría que $A \succ C$. El argumento para defender la condición de transitividad es que un agente que no respetase la transitividad en sus preferencias se comportaría de modo irracional. Supóngase, por ejemplo, que el agente tiene las preferencias no transitivas $A \succ B \succ C \succ A$, donde A, B y C representan bienes que pueden ser libremente intercambiados. Si el agente tiene en este momento el bien A, se le puede ofertar C y pedirle cierta cantidad de dinero por A. Dado que el agente prefiere tener el bien C, de buena gana podría darnos cierta cantidad de dinero para realizar este intercambio. De la misma forma, se le podría ofertar B por C, consiguiendo más dinero, y finalmente venderle A por B. De esta forma, se volvería al estado inicial, si bien ahora el agente dispone de menos dinero (Figura 16.1(a)). Además, podría reiterarse este ciclo hasta dejar al agente sin dinero. Claramente en este caso, el agente no actúa de forma racional.

Las seis restricciones siguientes se conocen como los axiomas de la teoría de la utilidad. Especifican las restricciones semánticas más obvias sobre las relaciones de preferencia y las loterías.

ORDENACIÓN

- **Ordenación**: dados dos estados cualesquiera, un agente racional debe ser capaz de, o bien preferir uno de ellos, o bien establecer ambos igualmente preferibles.

Figura 16.1 (a) Un ciclo de intercambios mostrando que las preferencias no transitivas A $\succ$ B $\succ$ C $\succ$ A derivan en un comportamiento irracional. (b) El axioma de descomposición.

Es decir, el agente no puede evitar tomar una decisión. Como se comentó en el Apartado 13.3, negarse a apostar es como negarse a que el tiempo pase.

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

TRANSITIVIDAD

- **Transitividad**: dados tres estados cualesquiera, si un agente prefiere A frente a B y a B sobre C, entonces el agente debe preferir A sobre C.

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

CONTINUIDAD

- **Continuidad**: si existe algún estado B entre A y C en la relación de preferencia, entonces existe un valor de probabilidad p para el que al agente racional le es indiferente entre considerar como seguro el estado B y la lotería que establece el estado A con probabilidad p y al estado C con probabilidad $1 - p$.

$$A \succ B \succ C \Rightarrow \exists p\ [p, A;\ 1 - p, C] \sim B$$

SUSTITUCIÓN

- **Sustitución**: si un agente es indiferente entre dos loterías, A y B, entonces el agente es indiferente entre dos loterías más complejas en las que aparecen A y B, pero en una de ellas se sustituye B por A. Esto es así, sin importar las probabilidades involucradas, ni el resto de resultados posibles en las loterías.

$$A \sim B \Rightarrow [p, A;\ 1 - p, C] \sim [p, B;\ 1 - p, C]$$

MONOTONICIDAD

- **Monotonicidad**: supóngase que existen dos loterías que tienen dos resultados posibles e iguales, A y B. Si un agente prefiere a A frente a B, entonces el agente debe preferir la lotería que presenta una mayor probabilidad para A (y viceversa).

$$A \succ B \Rightarrow (\ p \geq q \Leftrightarrow [p, A;\ 1 - p, B] \succsim [q, A;\ 1 - q, B])$$

DESCOMPOSICIÓN

- **Descomposición**: las loterías compuestas pueden reducirse a una forma más simple empleando las propiedades de la teoría de la probabilidad. A este axioma se le conoce como la regla de «no diversión en el juego» ya que establece que dos loterías consecutivas se pueden reducir en una única lotería equivalente, tal y como se muestra en la Figura 16.1(b)[1].

$$[p, A; 1 - p, [q, B; 1 - q, C]] \sim [p, A; (1 - p)q, B; (1 - p)(1 - q); C]$$

... y entonces apareció la utilidad

Conviene hacer notar que los axiomas de la teoría de la utilidad no dicen nada acerca de la utilidad, simplemente se refieren a las relaciones de preferencia. La preferencia se asume que es una propiedad básica de los agentes racionales. La existencia de una función de utilidad se *deriva* de los axiomas de utilidad:

1. **Principio de utilidad**
   Si las preferencias de un agente obedecen a los axiomas de utilidad entonces existe una función real U asociada a cada estado de forma que se tiene que $U(A) > U(B)$ si, y sólo si, se prefiere A frente a B, y $U(A) = U(B)$ si, y sólo si, el agente se muestra indiferente ante A o B.

$$U(A) > U(B) \Leftrightarrow A \succ B$$
$$U(A) = U(B) \Leftrightarrow A \sim B$$

2. **Principio de máxima utilidad esperada**
   La utilidad de una lotería es la suma de la probabilidad de cada resultado multiplicada por la utilidad de cada resultado.

$$U([p_1, S_1; \ldots; p_n, S_n)] = \sum_i p_i U(S_i)$$

En otras palabras, una vez que las probabilidades de ocurrencia y la utilidad de cada posible estado resultante están especificadas, la utilidad de la lotería que implica a tales estados queda determinada completamente. Dado que el resultado de una acción no determinista viene expresado por una lotería, esta expresión nos da la regla de decisión basada en el principio de MUE indicada en la Ecuación (16.1).

Es importante recordar que la existencia de una función que describe las preferencias en el comportamiento de un agente no significa, necesariamente, que el agente maximice explícitamente esta función de utilidad en sus propias deliberaciones. Tal y como se mostró en el Capítulo 2, el comportamiento racional puede generarse de múltiples formas, alguna de las cuales son más eficientes que la maximización explícita de la utilidad. Sin embargo, mediante la observación de las preferencias de un agente racional, es posible construir una función de utilidad que represente cuáles son los objetivos que, actualmente, persiguen las acciones del agente.

[1] Puede evaluarse la satisfacción de jugar mediante la codificación de los sucesos del juego en los estados de la descripción, por ejemplo, «Tener 10 dólares y haber apostado» puede ser preferible a «Tener 10 dólares y no haber apostado».

16.3 Funciones de utilidad

La utilidad es una función que establece una correspondencia entre los estados y el conjunto de los números reales. ¿Es esto todo lo que se puede decir sobre las funciones de utilidad? Hablando en sentido estricto, así es. Más allá de las restricciones enumeradas con anterioridad, un agente puede tener tantas preferencias como quiera. Por ejemplo, un agente puede preferir tener un número primo de dólares en su cuenta bancaria; en este caso, si él tuviera 16 dólares podría gastarse tres dólares. También podría preferir el modelo antiguo de Escarabajo de Volskwagen frente a un Mercedes último modelo. Además, las preferencias pueden interactuar: por ejemplo, el agente puede preferir sólo tener un número primo de dólares cuando tiene un Escarabajo y preferir tener más dinero cuanto tiene el Mercedes.

Si todas las funciones de utilidad fueran tan arbitrarias como las comentadas entonces la teoría de la utilidad no sería de mucha ayuda, ya que se tendrían que observar las preferencias del agente en cada una de las posibles combinaciones de circunstancias, antes de ser capaces de hacer cualquier predicción sobre su comportamiento. Afortunadamente, las preferencias de los agentes reales son, por lo general, más sistemáticas. Recíprocamente, existen formas sistemáticas de definir funciones de utilidad que, una vez implementadas en un agente artificial, pueden generar los tipos de comportamiento que se desean.

La utilidad del dinero

La teoría de la utilidad tiene sus raíces en la economía y la economía proporciona un candidato evidente para una medida de la utilidad: el dinero (más específicamente, el total de activos netos de un agente). La casi total aceptación del dinero como medio universal para intercambiar toda clase de bienes y servicios sugiere que el dinero representa un papel significativo en las funciones de utilidad humanas. (De hecho, mucha gente piensa que la *economía* es el estudio del dinero, mientras que los especialistas en economía se refieren más a la gestión, sobre todo, en lo que se refiere a las alternativas de inversión del dinero.)

Si se centra la atención a las acciones que exclusivamente afectan a la cantidad de dinero que tiene un agente, se pone de manifiesto que, por lo general, lo que el agente prefiere es tener más dinero, manteniendo igual el resto de recursos que posea. Se dice pues, que el agente presenta una **preferencia monótona** (siempre creciente) por el dinero. Sin embargo, esta condición no es suficiente para garantizar que el dinero se comporta como una función de utilidad ya que no se dice nada acerca de las preferencias sobre *loterías* que se refieren al dinero.

PREFERENCIA MONÓTONA

Suponga que es el ganador de un concurso de televisión y que el presentador le plantea una elección: o bien acepta directamente un premio de un millón de dólares o bien se juega el premio a cara o cruz, de forma que si sale cara, lo pierde todo, y si sale cruz, gana tres millones de dólares. Probablemente, como lo haría mucha más gente, usted desestimaría jugársela a cara o cruz y querría decir esto que usted es irracional.

VALOR MONETARIO ESPERADO

Suponiendo que usted confía en que la moneda no está trucada, el **valor monetario esperado** (VME) de la apuesta planteada es $\frac{1}{2}$ (0 \$) + $\frac{1}{2}$ (3.000.000 \$) = 1.500.000 \$, mientras que el valor VME en caso de aceptar el dinero directamente es de 1.000.000 de

dólares, que es menor. Pero fijarse en este criterio no implica que aceptar la apuesta sea la mejor opción posible. Suponga que por S_n, se denota la situación en que la cantidad de dinero total que dispone es de n dólares, y que actualmente usted tiene k dólares. Entonces las funciones de utilidad esperadas para cada posible elección en la apuesta son

$$EU\ (Aceptar) = \tfrac{1}{2}\, U(S_k) + \tfrac{1}{2} U(S_{k\,+\,3.000.000}),$$
$$EU\ (Rechazar) = U\ (S_{k\,+\,1.000.000})$$

Por lo tanto para decidir qué hacer, se necesita asignar valores de utilidad a los estados resultantes. La utilidad no es directamente proporcional al valor monetario ya que seguramente para usted la utilidad (una mejora en su nivel adquisitivo) del primer millón es mucho mayor que la utilidad de los millones adicionales. Suponga que asigna un valor de utilidad de 5 a su estado financiero actual (S_k), de 10 al estado $S_{k\,+\,3.000.000}$ y de 8 al estado $S_{k\,+\,1.000.000}$. Bajo estas condiciones, la decisión más razonable es rechazar la apuesta ya que la utilidad esperada, si se acepta, es de tan sólo 7,5 (menor que 8, el valor asociado a la decisión de no aceptar la apuesta). Suponga ahora que usted tiene 500 millones de dólares en su cuenta bancaria (y que se presentó al concurso simplemente para divertirse). En este caso, la apuesta será probablemente aceptable, ya que el beneficio adicional de tener 503 millones a tener 501 millones sería prácticamente similar.

En un estudio pionero sobre las funciones de utilidad actuales, Grayson (1960) encontró que la utilidad del dinero era casi exactamente proporcional al *logaritmo* de la cantidad de dinero; esta idea ya fue sugerida por Bernoulli (1738); (*véase* el Ejercicio 16.3). La Figura 16.2(a) muestra la curva de Beard que modela las preferencias establecidas por este autor y que resultan consistentes con la función de utilidad

$$U(S_{k\,+\,n}) = -263{,}31 + 22{,}09 \log(n + 150.000)$$

cuando n varía entre los valores de 150.000 dólares y 800.000 dólares.

No se debería contemplar aún a esta expresión como la función de utilidad definitiva para el valor monetario, pero es muy probable que para la mayoría de la gente la función de utilidad sea cóncava para valores positivos de la riqueza. Aunque endeudarse en

Figura 16.2 La utilidad del dinero. (a) Datos empíricos para la curva de Beard sobre un rango limitado. (b) Una curva para el rango completo.

exceso se puede considerar algo desastroso, las preferencias entre los diferentes niveles de endeudamiento pueden presentar un comportamiento invertido, en cuanto a la concavidad de la función de utilidad, es decir, tener una forma convexa. Por ejemplo, alguien que ya tenga una deuda de 10 millones de dólares puede asumir jugársela a cara o cruz si tiene posibilidad de ganar 10 millones si sale cara, o bien, pasar a perder 20 millones si sale cruz[2]. Este comportamiento puede representarse por una curva sigmoidal similar a la representada en la Figura 16.2(b).

Si se centra la atención al dominio positivo de la curva, donde la pendiente va decreciendo, entonces se tiene que, para cualquier lotería L, la utilidad de aceptar el reto es menor que la utilidad de recibir el valor monetario esperado de la lotería como algo seguro:

$$U(L) < U(S_{VME(L)})$$

MIEDO AL RIESGO

BÚSQUEDA DEL RIESGO

EQUIVALENTE DE CERTEZA

PRIMA DEL SEGURO

Es decir, los agentes que manejen una curva con esta forma sufrirán **miedo al riesgo**: ellos preferirán algo seguro, aunque tenga una retribución menor que el valor monetario esperado del reto. Por otro lado, en la región de desesperación o de valores muy negativos en la Figura 16.2(b), el comportamiento se caracteriza por **buscar el riesgo**. El valor que el agente estaría dispuesto a aceptar en lugar del resultado de la apuesta se conoce como **equivalente de certeza** o seguridad de la lotería. Algunos estudios han mostrado que la mayoría de la gente aceptaría unos 400 dólares en vez de jugárselo a una apuesta que le reportara 1.000 dólares la mitad de las veces y nada la otra mitad (es decir, el equivalente de certeza de la apuesta sería en este caso de 400 dólares). La diferencia entre el valor monetario esperado de la apuesta y su equivalente de certeza se conoce como **prima del seguro**. La aversión al riesgo es la base de la industria aseguradora ya que las primas de los seguros son siempre positivas. La gente prefiere pagar una pequeña prima de un seguro que arriesgar todo el valor de su vivienda en caso de incendio. Desde el punto de vista de la compañía aseguradora, el precio de la vivienda asegurada es muy pequeño en relación con sus activos totales. Esto se traduce en que la curva de utilidad del asegurador es prácticamente lineal en esta zona de valores pequeños y el riesgo asumido por la compañía es casi nulo.

NEUTRAL AL RIESGO

Conviene hacer notar que para *pequeños* cambios en la riqueza esperada frente a la actual, prácticamente cualquier curva se comporta linealmente. Un agente que tenga una curva lineal se dice que es **neutral al riesgo**. Para decisiones con un beneficio pequeño es de esperar adoptar una posición de neutralidad. En este sentido se justifica el procedimiento simplificado que proponía el empleo de pequeñas apuestas para evaluar probabilidades y para justificar los axiomas de la probabilidad en el Capítulo 13.

Escalas de utilidad y evaluación de la utilidad

Los axiomas de la teoría de la utilidad no especifican una única función de utilidad para un agente, conocidas sus preferencias. Por ejemplo, puede transformarse una función de utilidad $U(S)$ en otra como

$$U'(S) = k_1 + k_2\, U(S)$$

[2] Este comportamiento puede decirse que es desesperado, pero puede ser racional si uno se encuentra en una situación desesperada.

El Juicio Humano y la Falibilidad

La teoría de la decisión es una teoría **normativa**: describe cómo *debería* actuar un agente racional. Se mejoraría muchísimo en la aplicación de la teoría económica si además, la teoría de la decisión fuera **descriptiva**, es decir, reflejase cómo los seres humanos toman decisiones. Sin embargo, se tiene una indiscutible evidencia experimental que indica que la gente, sistemáticamente, viola los axiomas de la teoría de la utilidad. Los psicólogos Tverky y Kanheman (1982) proporcionan un claro ejemplo en este sentido que se basa en una situación planteada por el economista Allais (1953). A los participantes en el experimento se les planteó elegir entre las apuestas *A* y *B* y entre las apuestas *C* y *D*:

A: 80% probabilidad de ganar 4.000 $ *C*: 20% probabilidad de ganar 4.000 $

B: 100% probabilidad de ganar 3.000 $ *D*: 25% probabilidad de ganar 3.000 $

La mayoría de los participantes prefiri<ó la apuesta *B* frente a la *A*, y la *C* frente a la *D*. Si se considera que $U(0\ \$) = 0$, en la primera elección planteada se tiene que $0{,}8U(4.000\ \$) < U(3.000\ \$)$, en tanto que en la segunda elección ocurre lo contrario. Es decir, parece como si no existiese una función de utilidad que fuera consistente con las dos elecciones. Una posible conclusión sería decir, simplemente, que los seres humanos se comportan irracionalmente a juzgar por las normas de nuestros axiomas sobre la utilidad. Un punto de vista alternativo sería considerar que en el análisis realizado no se tuvo en cuenta el **arrepentimiento** (el sentimiento que un hombre sentiría cuando, sabiendo que cambia una recompensa segura (*B*) por otra que tiene un 80 por ciento de posibilidades de ocurrir, resulta que pierde). Es decir, si se elige la apuesta A, se tiene una probabilidad de un 20 por ciento de no conseguir nada y *sentirse como un completo idiota*.

Kahneman y Tversky continuaron con la idea de desarrollar una teoría descriptiva que explique cómo la gente huye del riesgo ante sucesos de una alta probabilidad de ocurrencia, pero están dispuestos a asumir mayores riesgos frente a situaciones poco probables. La relación entre este hecho evidente y la IA es que las decisiones que los agentes pueden adoptar son sólo tan buenas como las preferencias en las que están basadas. Si los informadores humanos insisten en juicios de preferencia contradictorios, no hay nada que un agente pueda hacer que sea consistente con ellos.

Afortunadamente, los juicios de preferencia hechos por los hombres están sujetos, a menudo, a revisiones a la vista de nuevas consideraciones. En uno de los primeros trabajos sobre la evaluación de la utilidad del dinero en la Harvard Business School, Keeney y Raiffa (1976, página 210) afirmaban lo siguiente:

> Muchas de las investigaciones empíricas han demostrado que hay serias deficiencias en el protocolo de valoración. Los sujetos tienden a tener demasiada aversión al riesgo en las situaciones pequeñas y consecuentemente... la funciones de utilidad presentan indebidamente grandes primas de riesgo para apuestas con una gran variabilidad... Sin embargo, la mayoría de los sujetos decisorios pueden reconciliar sus inconsistencias y sentir que han aprendido una lección importante acerca de cómo quieren comportarse. Como consecuencia, algunas personas deciden no renovar su seguro del automóvil a todo riesgo y deciden ampliar las primas de sus seguros de vida.

Incluso hoy en día, la racionalidad (irracionalidad) humana es objeto de una exhaustiva investigación.

donde k_1 es una constante y k_2 es cualquier constante *positiva*. Claramente, esta transformación lineal mantiene el comportamiento del agente inalterado.

En contextos *deterministas*, donde sólo es posible un único estado resultado y no varios, cualquier transformación *monótona* conserva el comportamiento inicial. Por ejemplo, puede calcularse la raíz cúbica de todas los valores de utilidad, sin que ello suponga un cambio en la ordenación de las preferencias. Se dice que un agente que se desenvuelve en un entorno determinista tiene una **función de utilidad ordinal** o **función de valor**; la función, más que aportar un valor numérico con un cierto significado, realmente proporciona una ordenación de los estados. Ya se vió esta distinción en el Capítulo 6 hablando de los juegos: las funciones de evaluación en juegos deterministas, como el ajedrez, son funciones de valor, mientras que las funciones de evaluación en juegos no deterministas como el *backgammon* son, verdaderamente, funciones de utilidad.

FUNCIÓN DE UTILIDAD ORDINAL

FUNCIÓN DE VALOR

Un procedimiento para valorar utilidades es establecer una escala que determine el «mejor premio posible» en el punto que se cumpla $U(S) = \text{SUP}(u)$, es decir, el valor mayor de u, y la «peor catástrofe posible» en $U(S) = \text{INF}(u)$. Los **valores de utilidad normalizados** emplean la escala en la cual $\text{INF}(u) = 0$ y $\text{SUP}(u) = 1$. Los valores de utilidad de resultados intermedios se establecen solicitando al agente que indique una preferencia entre un posible estado resultante S y una lotería estándar $[p, \text{INF}(u); (1 - p), \text{SUP}(u)]$. La probabilidad p se ajusta hasta que el agente se muestre indiferente frente a S y a la **lotería estándar**. Suponiendo que los valores de utilidad están normalizados, la utilidad de S viene dada por el valor de p.

UTILIDADES NORMALIZADAS

LOTERÍA ESTÁNDAR

En problemas de decisión relativos a la medicina, el transporte y el medio ambiente, entre otros, la vida de las personas es lo primero. En tales situaciones, $\text{INF}(u)$ es el valor que se asigna a una muerte inmediata (o quizás a muchas muertes). *Aunque a nadie le agrada poner un precio a una vida humana, es un hecho que estos balances se realizan continuamente*. Los aviones requieren una revisión completa que viene determinada por un cierto número de vuelos o de horas de vuelo, en vez de realizarla después de cada vuelo. Las carrocerías de los coches se hacen con planchas de metal relativamente delgadas para reducir costes, aun a costa de disminuir los índices de supervivencia en accidentes. Todavía se utilizan los carburantes con plomo, aun conociendo sus riesgos para la salud. Paradójicamente, una negativa a «invertir en vida» significa que la vida es a menudo infravalorada. Ross Shachter relata una experiencia con una agencia del gobierno norteamericano que encargó un estudio sobre la eliminación de un material de construcción que contiene amianto (asbesto) de las escuelas. El estudio consideró un cierto valor monetario para la vida de un niño en edad escolar, y defendía que era una opción razonable eliminar el asbesto. La agencia del gobierno, indignada, rechazó de inmediato el informe y se falló en contra de la eliminación del asbesto de las escuelas.

Ha habido algunos intentos de averiguar el lugar en el que las personas sitúan su propia vida. Dos de los criterios comunes, utilizados en el ámbito médico y de la seguridad, son la **micromortalidad** (probabilidad de que ocurra una muerte entre un millón) y la designada por las siglas **AVAC**, o Año de Vida Ajustado a la Calidad (equivalente a un año con buena salud, sin padecer ninguna enfermedad). Diferentes estudios sobre un amplio rango de individuos han mostrado que la micromortalidad está valorada sobre unos 20 dólares americanos (del año 1980). Ya se ha visto que las funciones no utilidad no tienen por qué ser lineales, y por lo tanto, esto no implica que un sujeto decisor

MICROMORTALIDAD

AVAC

pueda suicidarse por 20 millones de dólares. Otra vez, la linealidad local de cualquier curva de utilidad significa que tanto la micromortalidad como los valores AVAC son más adecuados en pequeños incrementos del riesgo y de la recompensa.

16.4 Funciones de utilidad multiatributo

En el ámbito de la administración pública, los factores a tener en cuenta en la toma de decisiones se refieren tanto a factores económicos como de salud y seguridad de los ciudadanos. Por ejemplo, cuando los responsables encargados de decidir qué niveles son admisibles en un entorno para una determinada sustancia cancerígena, éstos deben valorar el coste asociado a la prevención de las enfermedades ocasionadas por el agente cancerígeno y las pérdidas económicas que puede acarrear la supresión de determinados productos o procesos. El emplazamiento de un nuevo aeropuerto requiere considerar el impacto medioambiental que supone su construcción, el coste del suelo, la distancia de los centros urbanos, el ruido de las operaciones de vuelo, las condiciones de seguridad derivadas de la topografía y climatología local, etc. Los problemas de este tipo, caracterizados porque las decisiones dependen de dos o más atributos, se manejan mediante la **teoría de la utilidad multiatributo**.

TEORÍA DE LA UTILIDAD MULTIATRIBUTO

Se empleará la notación $\mathbf{X} = X_1, ..., X_n$ para denotar a un conjunto de atributos, y el vector $\mathbf{x} = \langle x_1, ..., x_n \rangle$ para denotar una asignación completa de valores a atributos. Por lo general, se entiende que cada atributo tiene un valor discreto o un valor escalar continuo. Para simplificar, se supondrá que cada atributo se define de tal forma, que un valor mayor para un atributo se corresponde con una mayor utilidad, siempre y cuando se consideren igual el resto de atributos. Por ejemplo, si se considera la *AusenciaDeRuido* como un atributo en el problema del emplazamiento del aeropuerto, entonces, conforme aumenta su valor, mejor será la solución. En algunos casos, puede ser necesario dividir el rango de valores de un atributo de forma que la utilidad crezca monótonamente dentro de cada rango.

A continuación se examinará en primer lugar una serie de casos en los que las decisiones deben adoptarse *sin* combinar los valores de los atributos en un único valor de utilidad. Después se verán casos en los que las utilidades de las combinaciones de atributos se pueden expresar claramente.

Predominio

Supóngase que el emplazamiento S_1 del aeropuerto requiere una inversión menor, genera una menor contaminación acústica y es más segura que el emplazamiento S_2. Uno no debería dudar en rechazar S_2. Se dice entonces que hay un **predominio estricto** de S_1 sobre S_2. En general, si una opción presenta valores más pequeños para todos los atributos que otra opción cualquiera, puede descartarse. A menudo, el criterio de predominio estricto es muy útil para reducir todas las opciones posibles a las que son realmente dignas de consideración, aunque rara vez se obtenga una única opción. La Figura 16.3(a) muestra un ejemplo para el caso de dos atributos.

PREDOMINIO ESTRICTO

Figura 16.3 Predominio estricto. (a) Determinista: opción A está estrictamente dominada por la opción B pero no por C ni D. (b) Incertidumbre: A está dominada estrictamente por B pero no por C.

Esto está muy bien para el caso determinista, donde los valores de los atributos se conocen con total certeza. ¿Qué pasa en el caso general, donde el resultado de una acción es incierto? Puede construirse una analogía directa con el criterio de predominio estricto donde, a pesar de la incertidumbre, todos los posibles resultados concretos de S_1 dominan estrictamente sobre todos los posibles resultados de S_2. (*Véase* Figura 16.3(b).) Por supuesto, es de esperar que esta condición se cumpla menos veces que en el caso determinista.

PREDOMINIO ESTOCÁSTICO

Afortunadamente, existe una generalización más útil, denominada **predominio estocástico**, que se da frecuentemente en las situaciones reales. El predominio estocástico es más sencillo de entender en el contexto de un único atributo. Supóngase que se cree que el coste de emplazar el aeropuerto en la localización S_1 se distribuye uniformemente entre 2.800 y 4.800 millones de dólares y que el coste de la localización S_2 se distribuye uniformemente entre 3.000 y 5.200 millones de dólares. La Figura 16.4(a) muestra estas distribuciones, donde el coste se representa con valores negativos. Entonces, conociendo sólo la información de que la utilidad decrece con el coste, podemos decir que S_1 domina de forma estocástica sobre S_2 (es decir, puede descartarse S_2). Es importante hacer notar que esta conclusión *no* se deriva de la comparación de los costes esperados. Por ejemplo, si se conociese que el coste de S_1 fue exactamente 3.800 millones entonces sería *imposible* tomar una decisión, sin información adicional sobre la utilidad del dinero[3].

La relación exacta entre las distribuciones de los atributos, necesarias para establecer el predominio estocástico, se aprecia mejor si se examinan las distribuciones de probabilidad *acumuladas*, tal y como se muestra en la Figura 16.4(b). La distribución de probabilidad acumulada mide la probabilidad de que el coste sea menor o

[3] Puede parecer extraño que tener *más* información sobre el coste de S_1 pueda hacer al agente *menos* capaz para decidir. La paradoja se resuelve si se considera que la decisión adoptada sin conocer el coste exacto es más improbable que sea la más correcta.

Figura 16.4 Predominio estocástico (a) S_1 domina estocásticamente a S_2 sobre el coste. (b) Distribuciones acumuladas para los costes negativos de S_1 y S_2.

igual a una determinada cantidad, es decir, es el resultado de la integración de la distribución de probabilidad original. Si la distribución de probabilidad acumulada de S_1 queda siempre a la derecha de la distribución de probabilidad acumulada de S_2 entonces, en términos estocásticos, el emplazamiento S_1 es más barato que el S_2. Formalmente, si dos acciones A_1 y A_2 siguen sendas distribuciones de probabilidad $p_1(x)$ y $p_2(x)$ respecto al atributo X, entonces A_1 domina estocásticamente sobre A_2 respecto al atributo X si

$$\forall x \quad \int_{-\infty}^{x} p_1(x')\, dx' \leq \int_{-\infty}^{x} p_2(x')\, dx'$$

La importancia de esta definición, en relación con la elección de las decisiones óptimas, se debe a la siguiente propiedad: *si A_1 domina estocásticamente a A_2 entonces para cualquier función de utilidad monótonamente no decreciente $U(x)$, la utilidad esperada de A_1 es, al menos, tan grande como la utilidad esperada de A_2*. Por lo tanto, si una acción es dominada estocásticamente por cualquier otra acción en todos los atributos entonces puede descartarse.

El predominio estocástico puede parecer un concepto demasiado técnico y que quizá no pueda evaluarse fácilmente sin un cálculo intensivo. En realidad, la condición de predominio estocástico puede establecerse fácilmente en muchos casos. Por ejemplo, supóngase que la construcción del aeropuerto depende de la distancia a los centros urbanos. El coste en sí es incierto, pero conforme aumenta la distancia, mayor es el coste. Si la localización S_1 es más cercana que la S_2, entonces S_1 dominará a S_2 en el coste asociado a la distancia. Aunque no se presentan aquí, existen algoritmos que permiten propagar este tipo de información cualitativa entre las variables de una **red probabilista cualitativa**, permitiendo a un sistema la toma de decisiones racional basándose en el criterio de predominio estocástico, sin necesidad de emplear valores numéricos.

RED PROBIBILISTA CUALITATIVA

Estructura de preferencia y utilidad multiatributo

Supóngase que se tienen n atributos, cada uno de los cuales tiene d valores posibles diferentes. Para especificar por completo la función de utilidad $U(x_1, ..., x_n)$ se necesitan d^n valores en el peor caso. Ahora, el peor caso se corresponde con la situación en la cual, las preferencias del agente no tienen una completa regularidad. La teoría de la utilidad multiatributo se fundamenta sobre la hipótesis de que las preferencias de los agentes típicos tienen mucha más estructura que el caso planteado. La aproximación básica es identificar regularidades en el comportamiento preferencial que se esperaría observar y utilizar lo que se denominan **teoremas de representación** para demostrar que un agente con una estructura de preferencia determinada tiene una función de utilidad de la forma

TEOREMAS DE REPRESENTACIÓN

$$U(x_1, ..., x_n) = f[f_1(x_1), ..., f_n(x_n)]$$

donde f es, por lo menos así se espera, una función más sencilla como por ejemplo la suma. *Véase* la similitud con el uso de las redes Bayesianas para descomponer la distribución de probabilidad conjunta de varias variables aleatorias.

Preferencias sin incertidumbre

Se comenzará con el caso determinista. Conviene recordar que para entornos deterministas el agente tiene una función de valor $V(x_1, ..., x_n)$ y que ahora se trata de representar esta función de forma concisa. La regularidad básica que aparece en las estructuras deterministas de preferencia se denomina **independencia de preferencia**. Dos atributos X_1 y X_2 son independientes en cuanto a la preferencia de un tercer atributo X_3, si la preferencia entre los resultados no depende del valor particular x_3 para el atributo X_3.

INDEPENDENCIA DE PREFERENCIA

Retomando el ejemplo del aeropuerto, donde se tienen (entre otros atributos) el *Ruido*, el *Coste* y la *Siniestralidad*, alguien puede proponer que el *Ruido* y el *Coste* son independientes en cuanto a la preferencia del atributo *Siniestralidad*. Por ejemplo, si se prefiere una localización con un coste de 4.000 millones de dólares y con una población de 20.000 personas bajo el corredor de aproximación al aeropuerto, frente a una localización con un coste de 3.700 millones y una población de 70.000 bajo el corredor de aproximación, cuando el índice de siniestralidad es, en ambos casos, de 0,06 muertes por millón de pasajeros, entonces se mantendrá esta preferencia cuando el índice de siniestralidad sea de 0,13 o de 0,01; y similar independencia se mantendría para preferencias entre cualquier otro par de valores para los atributos *Ruido* y *Coste*. Aparentemente, los atributos *Coste* y *Siniestralidad* también son independientes en cuanto a la preferencia del atributo *Ruido*, así como el *Ruido* y la *Siniestralidad* frente al *Coste*. Se dice que el conjunto de atributos {*Ruido*, *Coste*, *Siniestralidad*} presenta una **independencia mutua de preferencia** (IMP). La propiedad IMP viene a decir que si bien cualquier atributo puede ser importante, este hecho no afecta al modo en que alguien valore los restantes atributos frente a cada atributo.

INDEPENDENCIA MUTUA DE PREFERENCIA

A partir de la noción de independencia mutua de preferencia y gracias a un notable resultado demostrado por el economista Debreu (1960), puede describirse la función de valor de un agente de un modo muy sencillo: *Si los atributos $X_1, ..., X_n$ son mutuamen-*

te independientes en cuanto a la preferencia, entonces el comportamiento preferencial del agente puede describirse mediante la maximización de la función

$$V(x_1, \ldots, x_n) = \sum_i V_i(x_i),$$

donde cada función de valor V_i se refiere exclusivamente al atributo X_i. Por ejemplo, puede darse por bueno que en el caso del aeropuerto el criterio de decisión viene dado por la función de valor

$$V\,(ruido, coste, siniestralidad) = -ruido \times 10^4 - coste - siniestralidad \times 10^{12}.$$

FUNCIÓN DE VALOR ADITIVA

Una función de valor de este tipo se dice que es una **función de valor aditiva**. Las funciones aditivas constituyen una forma muy natural de describir la función de valor de un agente y son válidas en muchas situaciones del mundo real. Incluso cuando la propiedad de independencia mutua no se cumple estrictamente, como puede ser para el caso de los valores extremos de los atributos, una función de valor aditiva puede proporcionar una buena aproximación de las preferencias del agente. Esto es especialmente cierto cuando las violaciones de la propiedad de independencia mutua se dan en aquellas regiones de los rangos de los atributos que son muy improbables de ocurrir en la práctica.

Preferencias con incertidumbre

Cuando en un dominio está presente la incertidumbre, se necesita tanto contemplar la estructura de las relaciones de preferencia entre loterías como entender las propiedades resultantes de las funciones de utilidad. Las herramientas matemáticas para manejar esta situación son bastante complicadas, por lo que se presentan los resultados más importantes para tener una idea de lo que puede hacerse en este caso. Para tener una visión más extensa de este campo se remite al lector a consultar el trabajo de Keeney y Raiffa (1976).

INDEPENDENCIA DE UTILIDAD

La noción básica de **independencia de utilidad** amplía la noción de independencia de preferencia para cubrir el caso de las loterías. Un conjunto de atributos **X** es independiente en cuanto a la utilidad respecto a un conjunto de atributos **Y** si las preferencias entre loterías sobre los atributos de **X** son independientes de los valores particulares de los atributos de **Y**. Un conjunto de atributos cumple la condición de **independencia mutua de utilidad** (IMU) si cada uno de sus subconjuntos es independiente en cuanto a la utilidad con respecto al resto de atributos. Una vez más, parece razonable considerar que los atributos del ejemplo del aeropuerto cumplen la condición de independencia de utilidad mutua.

INDEPENDENCIA MUTUA DE UTILIDAD

FUNCIÓN DE UTILIDAD MULTIPLICATIVA

La propiedad IMU implica que el comportamiento del agente se puede describir empleando una **función de utilidad multiplicativa** (Keeney, 1974). La forma general de una función multiplicativa se ve mejor si se considera el caso de tres atributos. Para simplificar la notación, se empleará la notación U_i para designar a $U_i(x_i)$:

$$U = k_1U_1 + k_2U_2 + k_3U_3 + k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_3k_1U_3U_1 + k_1k_2k_3U_1U_2U_3$$

Aunque esta forma no parece muy sencilla, sólo contiene tres funciones de utilidad sobre un único atributo y tres constantes. Por lo general, un problema con n atributos que

verifique la propiedad IMU puede modelarse empleando *n* funciones de utilidad sobre un único atributo y *n* constantes. Cada una de estas funciones de utilidad puede desarrollarse sin considerar el resto de atributos, y mediante su combinación multiplicativa estará garantizada la correcta generación de las preferencias globales. Para obtener un modelo aditivo es necesario contemplar una serie de hipótesis adicionales.

16.5 Redes de decisión

DIAGRAMA DE INFLUENCIAS

REDES DE DECISIÓN

En esta sección, se examinará un mecanismo general cuyo propósito es la toma de decisiones de forma racional. Aunque en la bibliografía se denomina a menudo a este mecanismo como **diagrama de influencias** (Howard y Matheson, 1984), en este texto se empleará el término **redes de decisión** por considerarlo más descriptivo. Las redes de decisión combinan las redes Bayesianas con dos tipos de nodos adicionales para expresar acciones y utilidades. Se utilizará como ejemplo el problema del emplazamiento del aeropuerto.

Representación de un problema de decisión mediante una red de decisión

En su forma más general, una red de decisión representa información sobre el estado actual del agente, sus posibles acciones, el estado que resultará de la acción del agente y la utilidad del estado resultante. Por lo tanto, este formalismo proporciona una forma de implementar agentes basados en utilidad, introducidos por primera vez en la Sección 2.4. La Figura 16.5 muestra una red de decisión para el problema del emplazamiento del aeropuerto. En ella aparecen los tres tipos de nodos que se utilizan en una red de este tipo:

NODOS ALEATORIOS

- **Nodos aleatorios** (elipses) representan variables aleatorias y se interpretan igual que los nodos de una red Bayesiana. El agente puede tener incertidumbre acerca del coste de construcción, la densidad de tráfico aéreo y la posibilidad de plantear recursos legales. También las variables *Siniestralidad*, *Ruido* y *Coste* total son inciertas y cada una de ellas depende del emplazamiento elegido. Cada nodo aleatorio tiene asociada una distribución de probabilidad condicionada para cada combinación de valores de sus padres. En las redes de decisión, pueden ser padre de un nodo aleatorio tanto otro nodo aleatorio como un nodo de decisión. Nótese que cada nodo aleatorio del estado actual puede formar parte de una red de Bayes mayor, empleada para evaluar los costes de construcción, la densidad de tráfico aéreo y la posibilidad de plantear recursos legales.

NODOS DE DECISIÓN

- **Nodos de decisión** (rectángulos) representan puntos donde el sujeto decisor tiene que elegir una de las acciones posibles. En este caso, la acción de *Emplazar* puede tomar un valor diferente para cada emplazamiento bajo consideración. La elección influye sobre el coste, la seguridad y la contaminación acústica resultante. El Capítulo 17 trata con los casos en los que se debe adoptar más de una decisión.

Figura 16.5 Una red de decisión sencilla para el problema del emplazamiento del aeropuerto.

NODOS DE UTILIDAD

- **Nodos de utilidad** (rombos) representan las funciones de utilidad del agente[4]. El nodo de utilidad tiene como padres a todas aquellas variables descriptivas que influyen directamente sobre el valor de utilidad. Asociado con cada nodo de utilidad, se tiene una descripción de la utilidad del agente como una función de los atributos padre. La descripción puede ser tanto una representación tabular de la función como una función parametrizada de tipo aditivo o multiplicativo.

También en muchos casos se utiliza una forma simplificada. La notación no se modifica, pero los nodos aleatorios intermedios, aquellos que describen el estado resultante, se omiten. En su lugar, el nodo de utilidad se conecta directamente a los nodos con el estado actual y el nodo de decisión. En este caso, más que representar una función de utilidad sobre los estados, el nodo de utilidad representa la utilidad *esperada* y asociada a cada acción, tal y como se ha definido en la Ecuación (16.1). Por lo tanto, se denominarán a estas tablas por **tablas de acción-utilidad**. La Figura 16.6 ilustra la representación en la forma acción-utilidad del problema del aeropuerto.

TABLAS DE ACCIÓN-UTILIDAD

Nótese que, dado que los nodos *Ruido*, *Siniestralidad* y *Coste* que aparecen en la Figura 16.5 se refieren a estados futuros, nunca se pueden establecer sus valores como las variables que aportan evidencia. Por lo tanto, la versión simplificada (que omite a estos nodos) puede usarse incluso cuando puede emplearse también la forma más general. A pesar de que la forma simplificada contiene menos nodos, la omisión de una descripción explícita del resultado de la decisión del emplazamiento se traduce en que la red es menos flexible respecto a los cambios en las circunstancias. Por ejemplo, en la Figura 16.5, un cambio en los niveles de ruido de los aviones puede reflejarse en la red mediante un cambio en las tablas de probabilidad condicionada asociada con el nodo

[4] Estos nodos se denominan a menudo nodos de valor en la bibliografía. En este texto se prefiere mantener la distinción entre funciones de valor y de utilidad puesto que, tal y como ya se ha discutido con anterioridad, el estado resultante puede representar una lotería.

Figura 16.6 Una representación simplificada del problema del emplazamiento del aeropuerto. Se han eliminado los nodos aleatorios correspondientes a los estados resultantes.

Ruido, mientras que un cambio en el factor de ponderación del factor contaminación acústica en la función de utilidad se puede reflejar mediante un cambio en la tabla de utilidad. Por otra parte, en el diagrama acción-utilidad (Figura 16.6) todos los cambios indicados deben reflejarse por cambios en la tabla de acción-utilidad. Esencialmente, la formulación acción-utilidad es una versión *resumida* de la formulación original.

Evaluación en redes de decisión

Las acciones se seleccionan mediante la evaluación de la red de decisión para cada posible combinación de los valores de decisión. Una vez fijada la decisión, el nodo de decisión se comporta exactamente igual que un nodo aleatorio que haya sido configurado como una variable que aporta evidencia. El algoritmo para evaluar las redes de decisión es el siguiente:

1. Establecer las variables de evidencia para el estado actual.
2. Para cada valor posible del nodo de decisión:
   a) Establecer el nodo de decisión a este valor.
   b) Calcular las probabilidades a posteriori de los nodos padre del nodo de utilidad, empleando el algoritmo estándar de inferencia probabilista.
   c) Calcular la utilidad resultante para la acción.
3. Devolver la acción con el mayor valor de utilidad.

Ésta es una extensión directa del algoritmo de inferencia empleado en redes Bayesianas y puede incorporarse fácilmente en el diseño de un agente dado por la Figura 13.1. En el Capítulo 17 se verá que la posibilidad de ejecutar varias acciones en secuencia hace mucho más interesante el problema.

16.6 El valor de la información

En el análisis precedente, hemos supuesto que toda la información es relevante, o al menos que se proporciona al agente toda la información disponible antes de adoptar su decisión. En la práctica, casi nunca éste es el caso. *Uno de los aspectos más importantes en la toma de decisiones es el conocimiento de qué preguntas hacer*. Por ejemplo, un médico no puede esperar que se le proporcione el resultado de *todas las posibles* pruebas diagnósticas e interroga al paciente cuando acude a su consulta por primera vez[5]. Las pruebas, a menudo, son muy costosas y algunas veces incluso peligrosas (tanto por las pruebas en sí, como por el retraso que introducen). Su importancia depende de dos factores: si el resultado de la prueba podría contribuir significativamente a una mejora del tratamiento y qué probables son los resultados de las diferentes pruebas.

TEORÍA DEL VALOR DE LA INFORMACIÓN

Esta sección presenta la **teoría del valor de la información**, que capacita a un agente para seleccionar qué información adquirir. La adquisición de información se consigue mediante **acciones de percepción**, tal y como se describió en el Capítulo 12. Dado que la función de utilidad del agente rara vez se refiere a los contenidos del estado interno del agente, en tanto que el propósito general de las acciones sensitivas es modificar el estado interno, debemos evaluar las acciones sensitivas mediante sus efectos sobre las consiguientes acciones «reales» del agente. Por lo tanto, la teoría del valor de la información implica una forma de toma de decisiones secuenciales.

Un ejemplo sencillo

Supóngase que una compañía petrolera pretende adquirir uno de los n bloques indistinguibles de derechos de perforaciónes oceánicas. Supóngase además, que sólo uno de los bloques contiene un valor petrolífero de C dólares y que el precio de cada bloque es de C/n dólares. Si la actitud de la compañía hacia el riesgo es neutral entonces será indiferente ante la decisión de comprar o no comprar el bloque.

Ahora, supóngase que un geólogo ofrece a la compañía los resultados de un estudio sobre el bloque número 3, que indica, de forma definitiva, si en ese bloque hay o no petróleo. ¿Cuánto dinero debería estar dispuesta a pagar la compañía por esa información? La forma de responder a esta pregunta depende de lo que podría hacer la compañía con esta información:

- Con una probabilidad $1/n$, el estudio indicará la presencia de petróleo en el bloque 3. En este caso, la compañía comprará el bloque 3 por C/n dólares y obtendrá un beneficio de $C - C/n = (n - 1)C/n$ dólares.
- Con una probabilidad de $(n - 1)/n$, el estudio mostrará que en el bloque no hay petróleo, en cuyo caso la compañía comprará un bloque diferente. La probabilidad de encontrar petróleo en uno de los otros bloques pasa de $1/n$ a $1/(n - 1)$, por lo que la compañía puede evaluar el beneficio esperado en $C/(n - 1) - C/n = C/n(n - 1)$ dólares.

[5] En Estados Unidos, la única pregunta que se hace siempre antes de pasar consulta es si el paciente está asegurado o no.

Ahora puede calcularse el beneficio esperado, conocido el resultado del informe, mediante la expresión:

$$\frac{1}{n} \times \frac{(n-1)C}{n} + \frac{n-1}{n} \times \frac{C}{n(n-1)} = C/n$$

Por lo tanto, la compañía debería estar dispuesta a pagar al geólogo hasta C/n dólares por la información: la información vale tanto como el bloque mismo.

El valor de la información se deriva del hecho de que *con* la información, el curso de la acción puede modificarse para adaptarse a la situación *actual*. Por lo tanto, se puede discriminar de acuerdo a la situación, mientras que sin información, uno tiene que ver cuál es la mejor opción mediante el promedio sobre todas las posibles situaciones. En general, el valor de una información particular dada, se define como la diferencia entre el valor esperado de las mejores acciones, antes y después de disponer de la información.

Una fórmula general

VALOR DE LA INFORMACIÓN PERFECTA

Es sencillo derivar una fórmula matemática y general para el valor de la información. Habitualmente, se supone la obtención de una evidencia exacta acerca de una variable aleatoria E_j, de forma que se utiliza la expresión **valor de la información perfecta** (VIP)[6]. Sea E el conocimiento actual de un agente. Entonces el valor de la mejor acción a para el estado actual α se define mediante

$$EU(\alpha|E) = \max_A \sum_i U(Resultado_i(A))\, P(Resultado_i(A)|Realizar(A), E)$$

y el valor de la mejor acción a la vista de una nueva evidencia E_j será

$$EU(\alpha_{E_j}|E, E_j) = \max_A \sum_i U(Resultado_i(A))\, P(Resultado_i(A)|Realizar(A), E, E_j)$$

Pero si E_j es una variable aleatoria cuyo valor *actual* se desconoce, entonces debe promediarse sobre todos los posibles valores e_{jk} que puede tomar E_j, empleándose la creencia *actual* sobre su valor. El valor de determinar E_j, dado el estado actual de información E, se define como

$$VIP_E(E_j) = \left(\sum_k P(E_j = e_{jk}|E) EU(\alpha_{e_{jk}}|E, E_j = e_{jk})\right) - EU(\alpha|E)$$

Con el fin de interpretar intuitivamente esta fórmula, considérese el caso sencillo en que sólo son posibles dos acciones A_1 y A_2, entre las cuales se debe elegir una. Sus valores de utilidad actuales son U_1 y U_2, respectivamente. La información E_j hará que los nuevos valores esperados de utilidad sean U'_1 y U'_2 para cada acción, pero, antes de obtener E_j se tiene cierta distribución de probabilidad sobre todos los posibles valores de U'_1 y U'_2 (que se supondrán independientes).

[6] La información imperfecta acerca de la variable X se puede modelar como información perfecta sobre otra variable Y que está relacionada de forma probabilista con X. El Ejercicio 16.11 ilustra esta relación mediante un ejemplo.

Supóngase que A_1 y A_2 representan dos rutas diferentes a través de una cadena montañosa en invierno. A_1 es una carretera buena y sin muchas curvas que pasa por un puerto de montaña no muy alto y A_2 es una carretera de tierra y sinuosa que pasa por la cima de la montaña. Una vez conocida esta información, la ruta A_1 es claramente preferible, ya que es bastante probable que la segunda ruta se encuentre bloqueada por las avalanchas, mientras que es bastante improbable que la primera se encuentre bloqueada por el tráfico. Por lo tanto, el valor de utilidad U_1 es claramente mayor que U_2. Es posible obtener informes de satélites E_j sobre el estado actual de cada carretera que podría dar nuevos valores de utilidad esperados, y, para los dos trayectos. Las distribuciones de estos valores esperados se muestran en la Figura 16.7(a). Obviamente, en este caso, no merece la pena obtener los informes del satélite ya que es improbable que la información derivada de ellos cambie nuestro plan. Si no hay cambio, la información no tiene valor.

Supóngase ahora que se está planteando la posibilidad de elegir una de entre dos carreteras de tierra, sinuosas y con una longitud similar y que se transporta a un pasajero gravemente herido. En ese caso, incluso cuando U_1 y U_2 están bastante próximas, las distribuciones de U_1' y U_2' son muy amplias. Existe una posibilidad significativa de que la segunda ruta pase a estar despejada mientras que la primera se encuentre bloqueada y, en este caso, la diferencia en los valores de utilidad será elevada. El valor de la información perfecta indica que puede merecer la pena conseguir los informes del satélite. Tal situación se muestra en la Figura 16.7(b).

Otra situación diferente se da cuando se tiene que elegir entre las dos carreteras de tierra para hacer el viaje en verano, cuando las avalanchas de nieve son improbables. En este caso, los informes del satélite pueden mostrar que una de las rutas sea más bonita que otra por atravesar praderas alpinas florecientes, o quizá más peligrosa a causa de las tormentas. Es bastante probable que se cambiasen los planes si se contara con la información. Pero en este caso, es muy probable que la diferencia numérica entre las dos rutas sea muy pequeña, por lo que no se tomarán molestias en obtener los informes. Esta situación se muestra en la Figura 16.7(c).

Figura 16.7 Tres casos genéricos de valor de la información. En (a) A_1 seguirá siendo casi con toda seguridad superior a A_2 por lo que no es necesaria la información. En (b), la elección no está clara y la información es crucial. En (c), la elección no está clara pero puede haber una pequeña diferencia por lo que la información tiene menos valor.

En definitiva, puede afirmarse que *la información tiene valor en la medida en que tiene probabilidad de provocar la modificación de un plan y en la medida en que el nuevo plan resulte significativamente mejor que el anterior*.

Propiedades del valor de la información

Uno puede preguntarse si es posible que la información sea perjudicial: ¿puede tener un valor esperado negativo? Intuitivamente, debería esperarse que esta situación fuera imposible. Después de todo, puede ignorarse la información en el peor caso y aparentar que nunca se ha recibido. Este hecho es confirmado por el teorema siguiente que es aplicable a cualquier agente basado en la teoría de la decisión:

El valor de la información es no negativo:

$$\forall\, j, E \quad VIP_E(E_j) \geq 0$$

El teorema se deriva directamente de la definición del término VIP y se deja la prueba como ejercicio (Ejercicio 16.12). Es importante recordar que el término VIP depende del estado actual de información, y de ahí, la presencia del subíndice en el enunciado. Puede cambiar conforme se adquiere más información y en el caso extremo, puede llegar a ser nulo si la variable en cuestión tiene un valor conocido. Por lo tanto, el término VIP no es aditivo, es decir

$$VIP_E(E_j, E_k) \neq VIP_E(E_j) + VIP_E(E_k) \qquad \text{(en general)}$$

Sin embargo, el valor de la información perfecta es independiente del orden en que se obtengan las evidencias, lo cual parece, intuitivamente, obvio. Así, se tiene que

$$VIP_E(E_j, E_k) = VIP_E(E_j) + VIP_{E,E_j}(E_k) = VIP_E(E_k) + VIP_{E,E_k}(E_j)$$

La independencia del orden diferencia las acciones de percepción de las acciones ordinarias y simplifica el problema de determinar la secuencia en la que realizar las acciones de percepción.

Implementación de un agente recopilador de información

Un agente sensato debería hacer preguntas al usuario en un orden razonable, debería evitar preguntar cuestiones irrelevantes, debería considerar la importancia de cada información en función de su coste y debería parar de hacer preguntas cuando sea oportuno. Todas estas capacidades se pueden conseguir mediante la utilización del valor de la información como guía.

La Figura 16.8 muestra el diseño completo de un agente que puede recopilar información de forma inteligente, antes de actuar. Por ahora, se supondrá que con cada variable de evidencia observable E_j, se tiene un coste asociado, $Coste(E_j)$, que refleja el coste de obtener la evidencia a través de pruebas, consultores, cuestionarios o cualquier otra forma. El agente solicita aquella información que resulta más valiosa en comparación con su coste. Se entiende que el resultado de la acción $Solicitar(E_j)$ es tal, que la

```
función AGENTE-RECOPILADOR-INFORMACIÓN (percepción) devuelve una acción
  variables estáticas: D, una red de decisión

  integrar percepción dentro de D
  j ← valor que maximiza VIP(E_j) − Coste(E_j)
  si VIP(E_j) > Coste(E_j)
     entonces devolver SOLICITAR(E_j)
  si no devolver la mejor acción de D
```

Figura 16.8 Diseño de un sencillo agente recopilador de información. El agente funciona mediante la selección de la observación con el mayor valor de información de forma iterativa hasta que el coste de la siguiente observación es mayor que su beneficio esperado.

siguiente acción perceptiva proporciona el valor de E_j. Si ninguna acción perceptiva vale su coste, el agente selecciona una acción «real».

AGENTE MIOPE

El algoritmo descrito implementa una forma de recopilación de información que se denomina **miope**. Este calificativo se debe al hecho de que el agente utiliza el término VIP con poca visión de futuro, calculando el valor de la información como si fuera la única variable de evidencia a adquirir. Si no se tiene una única variable de evidencia que, de forma clara, ayude a adoptar la decisión, un agente miope puede adoptar precipitadamente una acción, cuando hubiera sido mejor solicitar dos o mas variables primero y luego adoptar la acción. El control miope se basa en el mismo criterio heurístico de la búsqueda voraz y a menudo funciona bien en la práctica (por ejemplo, esta estrategia ha demostrado ser mejor que expertos médicos en lo que se refiere a la elección de pruebas diagnósticas). Sin embargo, un agente recopilador de información racional ideal, debería considerar todas las posibles secuencias de solicitar información concluyendo con una acción externa y todas los posibles resultados de tales solicitudes. Dado que el valor de la segunda solicitud depende del resultado de la primera, el agente necesitaría explorar el espacio de **planes condicionados**, tal y como se describió en el Capítulo 12.

16.7 Sistemas expertos basados en la teoría de la decisión

ANÁLISIS DE LA DECISIÓN

El campo del **análisis de la decisión**, que se desarrolló en los años 50 y 60, estudia la aplicación de la teoría de la decisión a los problemas de decisión actuales. Se utiliza para ayudar a tomar decisiones razonables en dominios relevantes en los que están en juego intereses importantes, como son el mundo empresarial, gubernamental y legislativo, la estrategia militar, el diagnóstico médico y la salud pública, el diseño en la ingeniería y la gestión de recursos. El proceso incluye un estudio cuidadoso de todas las acciones posibles y sus consecuencias, así como las preferencias planteadas en cada resultado. En la terminología del análisis de la decisión es habitual distinguir entre el

SUJETO DECISOR

ANALISTA DE DECISIONES

sujeto decisor, que establece preferencias entre los resultados, y el **analista de decisiones** que enumera las acciones y resultados posibles y obtiene preferencias del sujeto que toma las decisiones para determinar el mejor curso de la acción. Hasta comienzos de la década de los 80, el objetivo principal del análisis de la decisión fue ayudar a los seres humanos en la toma de decisiones que realmente reflejen sus propias preferencias. Hoy en día, cada vez se automatizan más procesos de decisión y el análisis de la decisión se utiliza para garantizar que los procesos automáticos se comportan tal y como se desea.

Como se discutió en el Capítulo 14, las primeras investigaciones en el campo de los sistemas expertos se concentraron en responder preguntas, más que en tomar decisiones. Aquellos sistemas que no recomendaban acciones, más bien proporcionaban opiniones sobre diversos asuntos y, generalmente, lo hacían empleando reglas condición-acción, más que con representaciones explícitas de los resultados y las preferencias. La aparición de las redes Bayesianas a finales de los años 80 hizo posible la construcción de sistemas a gran escala que generaban inferencias probabilistas sólidas a partir de evidencias. La incorporación de las redes de decisión se traduce en que se pueden desarrollar sistemas expertos que recomienden decisiones óptimas, considerando tanto las preferencias del usuario como la evidencia disponible.

Un sistema que incorpore el tratamiento de valores de utilidad puede evitar una de las principales dificultades asociadas al proceso consultivo: la confusión entre probabilidad e importancia. Una estrategia corriente de los primeros sistemas expertos médicos, por ejemplo, fue ordenar los diagnósticos posibles en función de su probabilidad y considerar el más probable. Desafortunadamente, esto puede proporcionar resultados desastrosos. En la práctica, los dos diagnósticos más *probables* para la mayoría de los pacientes son habitualmente «que no padece nada serio» o «que tiene un resfriado», pero si para un paciente concreto, el tercer diagnóstico más probable es un cáncer de pulmón, se trata de un asunto grave. Obviamente, una prueba diagnóstica o un tratamiento debería depender tanto de la probabilidad como de la utilidad.

Se describirá el proceso de ingeniería del conocimiento para los sistemas expertos basados en la teoría de la decisión. Como ejemplo, se considerará el problema de seleccionar el tratamiento médico de una enfermedad cardíaca congénita en niños (*véase* Lucas, 1996).

COARTACIÓN DE LA AORTA

Cerca de un 0,8 por ciento de los niños nacen con una anomalía cardíaca, siendo la más común la **coartación de la aorta** (consistente en un estrechamiento de la aorta). Puede tratarse con cirugía, con una angioplastia (dilatación de la aorta mediante la introducción de un balón dentro de la arteria) o con medicación. El problema consiste en decidir qué tratamiento utilizar y cuándo hacerlo: cuanto más pequeño es el niño mayor es el riesgo de ciertos tratamientos, si bien no se puede esperar demasiado. Un sistema experto basado en la teoría de la decisión para este problema puede crearse partiendo de un equipo formado, al menos, por un experto del dominio (un cardiólogo pediatra) y un ingeniero del conocimiento. El proceso puede descomponerse en los pasos siguientes (que pueden compararse con los pasos seguidos en el desarrollo de sistemas basados en lógica mostrados en la Sección 8.4).

Crear un modelo causal. Determinar cuales son los posibles síntomas, afecciones, tratamientos y consecuencias. Una vez hecho esto dibujar arcos entre ellos, in-

dicando cuáles son las afecciones que causan determinados síntomas y qué tratamientos alivian las distintas afecciones. Algunas de estas relaciones serán conocidas por el experto del dominio y otras se extraerán de la bibliografía. A menudo, el modelo se ajustará bien a las descripciones gráficas informales que aparecen en los textos médicos.

Simplificar a un modelo de decisión cualitativo. Puesto que estamos utilizando el modelo para tomar decisiones acerca de los posibles tratamientos y no para otros fines (como determinar la distribución de probabilidad conjunta de determinadas combinaciones de síntomas/afecciones), puede simplificarse el modelo eliminando variables que no están involucradas en la determinación del tratamiento. Por ejemplo, el modelo original para el problema propuesto tendría una variable *Tratamiento* que toma como posibles valores *cirugía*, *angioplastia* o *medicación* y otra variable para la *Fecha* del tratamiento. Aún así, el experto puede tener dificultades para tratar por separado estas va-

Figura 16.9 Diagrama de influencia para el estrechamiento de la aorta (cortesía de Peter Lucas).

riables, y se pueden combinar de forma que, por ejemplo, *Tratamiento* pueda tomar valores tales como *cirugía en un mes*. Estas consideraciones se plasman en el modelo que aparece en la Figura 16.9.

Asignar probabilidades. Las probabilidades iniciales pueden establecerse a partir de bases de datos de pacientes, de estudios recogidos en la bibliografía o de las valoraciones subjetivas del experto. En los casos donde las probabilidades conocidas a través de la bibliografía no son condicionadas, pueden utilizarse técnicas como la regla de Bayes o la marginalización para obtener las probabilidades deseadas. Se ha constatado que los expertos son más capaces de valorar la probabilidad de un efecto dada una causa (por ejemplo, $P(disnea \mid fallo_cardíaco)$) que de cualquier otra forma.

Asignar utilidades. Cuando hay un número pequeño de resultados posibles, éstos pueden enumerarse y evaluarse individualmente. Se consideraría una escala desde el mejor al peor resultado asignando a cada extremo un valor numérico, por ejemplo, –1.000 en caso de fallecer el niño y 0 para una completa recuperación. Una vez hecho esto, deberían situarse el resto de resultados sobre la escala. Esto puede hacerlo el experto, pero es mejor si el paciente (o en el caso de niños, sus padres) está involucrado, ya que cada uno puede tener preferencias diferentes. Si los posibles resultados son excesivos, se necesitará alguna forma de combinarlos mediante funciones de utilidad multiatributo. Por ejemplo, puede establecerse que la utilidad negativa de varias complicaciones sea aditiva.

ESTÁNDAR DE ORO

Verificar y refinar el modelo. Para evaluar el sistema se necesitará un conjunto de pares (entrada, salida) correctos; el denominado **estándar de oro** con el que comparar. Para los sistemas expertos médicos esto significa a menudo reunir a los mejores especialistas, presentarles un conjunto reducido de casos y preguntarles por su diagnóstico y su tratamiento recomendado. Luego se trata de ver lo bien que el sistema se ajusta a sus recomendaciones. Si los resultados obtenidos son pobres, se tratará de aislar los componentes que funcionan incorrectamente y corregirlos. Puede resultar útil ejecutar el sistema «hacia atrás». En lugar de presentar al sistema los síntomas y preguntar por un diagnóstico, puede presentarse un diagnóstico como «fallo cardíaco», examinar la probabilidad pronosticada de síntomas como taquicardia y comparar con la bibliografía médica.

Realizar un análisis de la sensibilidad. Este paso importante comprueba si la mejor decisión es sensible a pequeños cambios en las probabilidades y utilidades asignadas mediante la variación sistemática de estos parámetros y la ejecución de la evaluación otra vez. Si cambios pequeños conducen a decisiones significativamente diferentes entonces podría merecer la pena gastar más recursos en obtener mejores datos. Si todas la variaciones conducen a la misma decisión, entonces el usuario tendrá una mayor confianza en que la decisión es la correcta. El análisis de la sensibilidad es particularmente importante ya que una de las principales críticas a la aproximación probabilista de los sistemas expertos es la excesiva dificultad para evaluar las probabilidades numéricas requeridas. El análisis de la sensibilidad revela a menudo que muchos de los valores necesitan especificarse sólo muy aproximadamente. Por ejemplo, puede tenerse incertidumbre acerca de la probabilidad previa $P(taquicardia)$, pero si se prueban muchos valores diferentes y en cada caso se obtiene la misma recomendación entonces uno se puede despreocupar un tanto de su ignorancia.

16.8 Resumen

Este capítulo muestra cómo combinar la teoría de la utilidad con la teoría de la probabilidad para permitir a un agente seleccionar acciones que maximicen su rendimiento esperado.

- La **teoría de la probabilidad** describe en qué debería creer un agente con base en la evidencia, la **teoría de la utilidad** describe lo que un agente desea, y la **teoría de la decisión** engloba a las dos para describir lo que debería hacer un agente.
- Puede utilizarse la teoría de la decisión para construir un sistema que adopte decisiones mediante la consideración de todas las acciones posibles y mediante la elección de aquella que conduce al mejor resultado esperado. Un sistema con estas características se conoce como un **agente racional**.
- La teoría de la utilidad demuestra que un agente cuyas preferencias entre loterías son consistentes con un conjunto de axiomas sencillos, puede describirse mediante una función de utilidad que el agente posee; además, el agente selecciona sus acciones mediante la maximización de su utilidad esperada.
- La **teoría de la utilidad multiatributo** trata con utilidades que dependen de varios atributos de los estados diferentes. El **predominio estocástico** es una técnica particularmente útil en la toma de decisiones ambiguas, incluso sin contar con los valores de utilidad precisos para los atributos.
- Las **redes de decisión** proporcionan un formalismo sencillo para expresar y resolver problemas de decisión. Son una extensión natural de las redes Bayesianas que incluyen nodos de decisión y de utilidad, además de los nodos aleatorios.
- En algunas ocasiones, la resolución de un problema implica encontrar más información antes de adoptar una decisión. El **valor de una información** se define como la mejora esperada en el valor de la utilidad comparándolo con la decisión tomada sin contar con tal información.
- Los **sistemas expertos** que incorporan información de utilidad presentan capacidades adicionales en comparación con los sistemas de inferencia puros. Además de ser capaces de tomar decisiones, pueden utilizar el valor de una información para decidir si la adquieren o no y pueden calcular la sensibilidad de sus decisiones frente a pequeños cambios en las asignaciones de probabilidad y utilidad.

Notas bibliográficas e históricas

Una de las primeras aplicaciones del principio de máxima utilidad esperada (aunque se trate de una variación que trata con utilidades infinitas) fue la Apuesta de Pascal, publicada por primera vez como parte de la *Port-Royal Logic* (Arnauld, 1662). Daniel Bernoulli (1738), investigando la paradoja de San Petersburgo (*véase* Ejercicio 16.3), fue el primero en comprender la importancia de evaluar preferencias para las loterías escribiendo «el *valor* de un artículo no debe basarse en su *precio* sino en la *utilidad* que proporciona» (cursivas propias del autor). Jeremy Bentham (1823) propuso el **cálculo**

hedonista para ponderar «placeres» y «molestias», arguyendo que todas las decisiones (no sólo las monetarias) pueden reducirse a comparaciones de utilidad.

El origen de las utilidades numéricas a partir de preferencias fue realizado por primera vez por Ramsey (1931); los axiomas para la preferencia en este texto son más próximas en su forma a las redescubiertas en el libro *Theory of Games and Economic Behavior* (von Neumann y Morgenstern, 1944). Una buena exposición de estos axiomas, en lo que se refiere a la discusión sobre la preferencia en el riesgo, se tiene en Howard (1977). Ramsey derivó las probabilidades subjetivas (no sólo utilidades) a partir de las preferencias de un agente; Savage (1954) y Jeffrey (1983) desarrollaron construcciones de este tipo más recientes. Von Winterfeldt y Edwards (1986) proporcionaron una perspectiva moderna al análisis de la decisión y su relación con las estructuras humanas de preferencia. La medida de la utilidad denominada micromortalidad se debate en Howard (1989). Un estudio de 1994 realizado por la publicación *The Economist* estableció el valor de una vida humana entre 750.000 dólares y 2,6 millones de dólares. Sin embargo, Richard Thaler (1992) encontró irracional la variación en el precio que uno está dispuesto a pagar para evitar un riesgo de fallecimiento, frente al precio que uno está dispuesto a cobrar por aceptar el riesgo. Para una probabilidad de uno entre 1.000, un encuestado no pagaría más de 200 dólares por evitar el riesgo pero no aceptaría 50.000 dólares por asumir el riesgo.

Los índices AVAC se utilizan más extensamente en problemas de decisión referentes a política social y de salud que los índices de micromortalidad: *véase* (Russel, 1999) para obtener un ejemplo típico de una argumentación de una importante decisión en el ámbito de la salud pública basada en un incremento de la utilidad esperada, medida en las unidades AVAC.

El libro *Decisions with Multiple Objectives: Preferences and Value Tradeoffs* (Keeney y Raiffa, 1976) proporciona una rigurosa introducción a la teoría de la utilidad multiatributo. Describe las primeras implementaciones computacionales de métodos para la obtención de los parámetros necesarios para una función de utilidad multiatributo e incluye una colección muy extensa de consideraciones en relación con aplicaciones reales de la teoría. En el ámbito de la IA, la referencia principal de la teoría de la utilidad multiatributo es un artículo de Wellman (1985) que incluye un sistema denominado URP (*Utility Reasoning Package*) que puede utilizar un conjunto de declaraciones sobre independencia de preferencia e independencia condicional para analizar la estructura de problemas de decisión. El uso del predominio estocástico junto con los modelos probabilistas cualitativos fueron intensamente investigados por Wellman (1988, 1990a). Wellman y Doyle (1992) proporcionan un esquema preliminar de cómo un conjunto complejo de relaciones utilidad-independencia pueden usarse para establecer un modelo estructurado de una función de utilidad del mismo modo que las redes Bayesianas aportan un modelo estructurado de las distribuciones de probabilidad conjuntas. Bacchus y Grove (1995, 1996) y La Mura y Shoham (1999) aportaron posteriores y más avanzados resultados en esta línea.

La teoría de la decisión ha sido una herramienta estándar en las ciencias económicas, empresariales y de gestión desde 1950. Hasta la década de los 80, los árboles de decisión fueron la herramienta principal que se utilizó para representar problemas de decisión sencillos. Smith (1988) proporciona una perspectiva general de la metodología del análisis de la decisión. Las redes de decisión o diagramas de influencia fueron introducidos por Howard y Matheson (1984) con base en un trabajo preliminar de un grupo (en el que es-

taban Howard y Matheson) en el SRI (Miller *et al*., 1976). El método de Howard y Matheson implicaba la derivación de un árbol de decisión a partir de una red de decisión, pero, por lo general, el tamaño del árbol crece rápidamente. Shachter (1986) desarrolló un método de toma de decisiones basado directamente sobre una red de decisión, sin necesidad de construir un árbol de decisión intermedio. Este algoritmo fue también uno de los primeros en proporcionar inferencia completa para varias redes Bayesianas conectadas. Un trabajo reciente de Nilsson y Lauritzen (2000) relaciona los algoritmos utilizados por las redes de decisión con desarrollos en curso de algoritmos de agrupamiento empleados por redes Bayesianas. La recopilación de Oliver y Smith (1990) proporciona una serie de artículos útiles sobre redes de decisión, al igual que el número especial del año 1990 de la revista *Networks*. Artículos sobre redes de decisión y modelado de funciones de utilidad aparecen regularmente en la revista *Management Science*.

Ron Howard (1966) fue el primero en analizar la teoría del valor de la información. Su artículo concluyó con la observación «Si la teoría del valor de la información y las estructuras asociadas a la teoría de la decisión no ocupan un lugar preponderante en la futura enseñanza de los ingenieros, entonces la ingeniería se encontrará con que su papel tradicional en la gestión científica y de los recursos económicos en pos del beneficio del hombre, habrá sido usurpada por otra profesión». Hasta la fecha no se ha dado la revolución esperada en los métodos de gestión, aunque la situación puede cambiar conforme se extienda el uso de la teoría del valor de la información en sistemas expertos Bayesianos.

Sorprendentemente, pocos investigadores de la IA adoptaron las herramientas de la teoría de la decisión después de las primeras aplicaciones en problemas de decisión médicos descritos en el Capítulo 13. Una de las pocas excepciones fue Jerry Feldman, quien aplicó la teoría de la decisión a problemas de visión (Feldman y Yakimovsky, 1974) y planificación (Feldman y Sproull, 1977). Después del resurgimiento del interés en los métodos probabilistas en la IA hacia la década de los 80, los sistemas expertos basados en la teoría de la decisión obtuvieron una amplia aceptación (Horvitz *et al*., 1988). De hecho, del año 1991 en adelante, el diseño de la portada de la revista *Artificial Intelligence* representa una red de decisión, si bien parece que se han tomado algunas licencias artísticas en cuanto a la dirección de los arcos.

EJERCICIOS

16.1 (Adaptado a partir del ejercicio propuesto por David Heckerman). Este ejercicio se refiere al **Juego del Almanaque** que utilizan los analistas de la decisión para calibrar estimaciones numéricas. Se trata de dar la mejor respuesta que uno puede intuir a cada una de las preguntas que se hacen, es decir, el número cuya probabilidad de ser demasiado elevado sea la misma que la de ser demasiado bajo. También estime el primer cuartil, es decir, el número que piense que tiene un 25 por ciento de probabilidades de ser demasiado elevado y un 75 por ciento de probabilidades de ser demasiado bajo. Haga lo mismo con el tercer cuartil. Una vez hecho esto, se tendrán tres estimaciones en total (baja, media y alta) para cada cuestión.

a) Número de pasajeros que volaron entre Nueva York y Los Ángeles en el año 1989.

b) La población de Varsovia en 1992.
c) El año en que Coronado descubrió el Río Mississippi.
d) El número de votos obtenidos por Jimmy Carter en las elecciones presidenciales de 1976.
e) La edad del árbol viviente más viejo.
f) La altura de la presa de Hoover (en pies).
g) El número de huevos producidos en Oregon en 1985.
h) El número de budistas en el mundo en el año 1992.
i) El número de muertes por SIDA en Estados Unidos en el año 1981.
j) El número de patentes concedidas en Estados Unidos en 1901.

Las respuestas correctas aparecen después del último ejercicio de este capítulo. Desde el punto de vista del análisis de la decisión, la cuestión importante no es cómo se aproximan sus estimaciones medias a los valores reales sino más bien la frecuencia en que la contestación real cae dentro de los límites estimados para el primer y tercer cuartil. Si está alrededor de la mitad de las veces, sus límites son precisos. Pero si, como la mayoría de la gente, usted está tan seguro de sí mismo de lo que debería, sus estimaciones caerán dentro del intervalo bastante menos veces que la mitad. Con la práctica, podrá calibrarse para dar unos límites más realistas y por lo tanto más útiles a la hora de aportar información en la toma de decisiones. Pruebe con este segundo juego de preguntas y compruebe si hay alguna mejora:

a) El año de nacimiento de Zsa Zsa Gabor.
b) La distancia máxima de Marte al sol (en millas).
c) El valor (en dólares) de las exportaciones de trigo de Estados Unidos en 1992.
d) Las toneladas manejadas en el puerto de Honolulu en el año 1991.
e) El salario anual (en dólares) del gobernador de California en 1993.
f) La población de San Diego en 1990.
g) El año en que Roger Williams fundó Providence, Rhode Island.
h) La altura del Monte Kilimanjaro (en pies).
i) La longitud del Puente de Brooklyn (en pies).
j) El número de muertes ocasionadas por accidentes de tráfico en Estados Unidos en 1992.

16.2 Los boletos de una lotería cuestan un dólar. Hay dos posibles premios: uno de 10 dólares con una probabilidad de uno entre 50 y otro de 1.000.000 dólares con una probabilidad de uno entre 2.000.000. ¿Cuál es el valor monetario esperado del boleto de lotería? ¿Cuándo es razonable comprar un boleto? Sea preciso (muestre una ecuación en la que se consideren valores de utilidad). Puede suponer que actualmente dispone de k dólares y que su utilidad es cero, $U(S_k) = 0$. También puede suponer que $U(S_{k+10}) = 10 \times U(S_{k+1})$, pero puede no hacer suposición alguna sobre $U(S_{k+1.000.000})$. Estudios sociológicos muestran que la gente con ingresos más pequeños compra un número desproporcionado de boletos de lotería. ¿Piensa que esto se debe a que son peores a la hora de tomar la decisión o a que tienen una función de utilidad diferente?

16.3 En 1738, J. Bernoulli investigó la paradoja de San Petersburgo que se describe a continuación. Usted tiene la oportunidad de jugar un juego en el que una moneda no tru-

cada se lanza repetidas veces hasta que salga cara. Si la primera cara aparece en el n-ésimo lanzamiento, ganará 2^n dólares.

a) Demuestre que el valor monetario esperado de este juego es infinito.
b) ¿Cuánto podría usted (personalmente) pagar por jugar a este juego?
c) Bernoulli resolvió la aparente paradoja mediante la sugerencia de que la utilidad del dinero se mide con una escala logarítmica (es decir, $U(S_n) = a \log_2 n + b$, donde S_n es el estado caracterizado por poseer n dólares). ¿Cuál es la utilidad esperada del juego bajo esta suposición?
d) ¿Cuál es la cantidad máxima que sería razonable pagar por jugar, suponiendo que la riqueza inicial es de k dólares?

16.4 Establezca su propia utilidad para diferentes incrementos de dinero mediante la ejecución de una serie de pruebas de preferencia entre una determinada cantidad M_1 y una lotería $[p, M_2; (1 - p), 0]$. Elija valores distintos para M_1 y M_2 y varíe p hasta que se muestre indiferente entre las dos opciones. Represente gráficamente la función de utilidad resultante.

16.5 Escriba un programa de computador que automatice el proceso indicado en el Ejercicio 16.4. Verifique los resultados del programa con varias personas de diferente poder adquisitivo e inclinaciones políticas. Comente la consistencia de sus resultados tanto para un individuo como entre individuos.

16.6 ¿Cuánto vale el índice de micromortalidad para usted? Conciba un protocolo para determinarlo. Pregunte por cuestiones basadas tanto en pagar por evitar un riesgo como en cobrar por asumirlo.

16.7 Demuestre que si X_1 y X_2 son independientes en cuanto a la preferencia de X_3, y que X_2 y X_3 son independientes respecto a X_1, entonces X_3 y X_1 son independientes en cuanto a la preferencia frente a X_2.

16.8 Este ejercicio completa el análisis del problema del emplazamiento del aeropuerto de la Figura 16.5.

a) Proporcione los recorridos de las variables, las probabilidades y utilidades para la red, considerando que son tres los posibles emplazamientos.
b) Resuelva el problema de decisión.
c) ¿Qué ocurre si un avance tecnológico hace que los aviones produzcan la mitad del ruido que ahora?
d) ¿Qué ocurre si la importancia de la prevención contra la contaminación acústica se multiplica por tres?
e) Calcule el valor de la información perfecta para *TráficoAéreo*, *Litigio* y *Construcción* para su modelo.

16.9 Repita el Ejercicio 16.8 empleando la representación mediante tablas acción-utilidad mostrada en la Figura 16.6.

16.10 Para cada uno de los diagramas del emplazamiento del aeropuerto de los Ejercicios 16.8 y 16.9, ¿qué entrada de la tabla de probabilidad condicionada es más sensible, dada la evidencia conocida?

16.11 (Adaptado a partir del ejercicio propuesto por Pearl (1988).) Un comprador de coches de segunda mano puede decidir realizar varias pruebas con diferentes costes (que pueden ir desde comprobar la presión de las ruedas con una patada hasta llevar el coche a un mecánico de confianza) y una vez realizadas, y dependiendo del resultado de las pruebas, decidir qué coche comprar. Supondremos que el comprador se encuentra decidiendo si compra el coche c_1, que tiene tiempo de realizar cuando menos una prueba y que t_1 es la prueba de c_1 y cuesta 50 dólares.

Un coche puede estar en buenas condiciones (calidad q^+) o en malas condiciones (calidad q^-) y las pruebas pueden ayudar a saber en qué estado se encuentra el coche. El coche c_1 cuesta 1.500 dólares y el valor de mercado es de 2.000 dólares si está en buenas condiciones. Si no es así, el coche necesitará una reparación cuyo coste ascenderá a 700 dólares. El comprador estima que hay un 70 por ciento de probabilidades de que el coche esté en buenas condiciones.

a) Dibuje la red de decisión que representa a este problema.

b) Calcule la ganancia neta estimada por la compra de c_1, suponiendo que no se realiza ninguna prueba.

c) Las pruebas se pueden describir mediante la probabilidad de que el coche pase o no el correspondiente chequeo, conociendo que el coche esté en buenas o malas condiciones. Tenemos la información siguiente:

$$P(superar(c_1, t_1) \mid q^+(c_1)) = 0{,}8$$
$$P(superar(c_1, t_1) \mid q^-(c_1)) = 0{,}35$$

Use el esquema de Bayes para calcular la probabilidad de que el coche pase (o falle) su test y por lo tanto la probabilidad de que está en buena (o mala) condición, dado cada posible resultado del test.

d) Calcule las decisiones óptimas conociendo que el coche supera o no la prueba y sus utilidades esperadas.

e) Calcule el valor de la información del chequeo y derive el plan condicionado óptimo para el comprador.

16.12 Pruebe que el valor de la información es no negativo e independiente del orden, tal y como se plantea en la Sección 16.6. Explique cómo, después de recibir una información, uno puede tomar una decisión peor que si la hubiera tomado antes de conocer tal información.

16.13 Modifique y amplíe el código de la red Bayesiana del repositorio de código para permitir crear y evaluar redes de decisión, así como, para calcular el valor de la información.

Las respuestas del Ejercicio 16.1 (donde M indica que el valor se multiplica por un millón) son: para el primer conjunto: 3M; 1,6M; 1.541; 41M; 4.768; 221; 649M; 295M; 132; 25.546. Para el segundo conjunto: 1917; 155M; 4.500M; 11M; 120.000; 1,1M; 1.636; 19.340; 1.595; 41.710.

17 Toma de decisiones complejas

Donde se examinan métodos para decidir qué hacer hoy, conociendo lo que podemos decidir de nuevo mañana.

PROBLEMAS DE DECISIÓN SECUENCIALES

Este capítulo trata sobre las cuestiones computacionales involucradas en la toma de decisiones. Mientras que el Capítulo 16 trató con problemas de decisión episódicos en los que la utilidad del resultado de cada acción es conocida, en este capítulo se tratan los **problemas de decisión secuenciales**, aquellos en los que la utilidad para el agente depende de la secuencia de acciones. Los problemas de decisión secuenciales que incluyen utilidades, incertidumbre y percepción, son una generalización de los problemas de búsqueda y planificación descritos en las Partes II y IV. La Sección 17.1 explica cómo se definen los problemas de decisión secuenciales y las Secciones 17.2 y 17.3 explican cómo pueden resolverse para obtener un comportamiento óptimo que equilibre los riesgos y las recompensas de actuar en un entorno incierto. La Sección 17.4 amplía estas ideas al caso de entornos parcialmente observables y la Sección 17.5 desarrolla un diseño completo para agentes basados en la teoría de la decisión en entornos parcialmente observables, que combinan las redes Bayesianas dinámicas del Capítulo 15 con las redes de decisión del Capítulo 16.

La segunda parte del capítulo cubre entornos con múltiples agentes. En tales entornos, la noción de comportamiento óptimo se complica mucho más debido a las interacciones entre agentes. La Sección 17.6 introduce las ideas principales de la **teoría de juegos**, incluyendo la idea de que los agentes racionales podrían necesitar comportarse aleatoriamente. La Sección 17.7 examina cómo pueden diseñarse sistemas multiagente de forma que múltiples agentes puedan alcanzar un objetivo común.

17.1 Problemas de decisión secuenciales

Un ejemplo

Supóngase que un agente se encuentra en el entorno de 4×3 celdas mostrado en la Figura 17.1(a). Comenzando en el estado inicial, debe elegir una acción en cada instante. La interacción con el entorno concluye cuando el agente alcanza uno de los estados objetivos, marcados con las etiquetas $+1$ o -1. En cada posición, las acciones disponibles se denominan *Arriba*, *Abajo*, *Izquierda* y *Derecha*. Se supondrá que, por el momento, el entorno es **completamente observable**, de forma que el agente siempre sabe dónde se encuentra.

Si el entorno fuera determinista, la solución sería fácil: [*Arriba*, *Arriba*, *Derecha*, *Derecha*, *Derecha*]. Desafortunadamente, el entorno no siempre dará con esta solución ya que las acciones no son fiables. El modelo concreto de movimiento estocástico adoptado se ilustra en la Figura 17.1(b). Cada acción alcanza el efecto deseado con una probabilidad de 0,8 pero, el resto de las veces, la acción mueve al agente en una dirección perpendicular a la deseada. Además, si el agente choca con una pared, permanece en la misma celda. Por ejemplo, partiendo de la celda inicial (1, 1), la acción *Arriba* mueve al agente a la posición (1, 2) con probabilidad 0,8, pero, con probabilidad 0,1 puede moverse a la derecha a la celda (2, 1), y con probabilidad 0,1 puede moverse a la izquierda, chocar con la pared, y permanecer en la posición (1, 1). En este entorno, la secuencia [*Arriba*, *Arriba*, *Derecha*, *Derecha*, *Derecha*] que trata de mover al agente siguiendo el contorno, alcanza el estado objetivo situado en (4, 3) con una probabilidad de $0{,}8^5 = 0{,}32768$. También existe una pequeña posibilidad de alcanzar accidentalmente el objetivo yendo por otro camino con una probabilidad $0{,}1^4 \times 0{,}8$ de modo que la probabilidad total global es de 0,32776. (*Véase* también el Ejercicio 17.1.)

Figura 17.1 (a) Un entorno sencillo de 4×3 que plantea al agente un problema de decisión secuencial. (b) Ilustración del modelo de transición del entorno: el resultado «deseado» ocurre con probabilidad 0,8, mientras que con probabilidad 0,2 el agente se mueve en las direcciones perpendiculares a la dirección deseada. Una colisión con la pared se traduce en la ausencia de movimiento. Los dos estados terminales tienen una recompensa de $+1$ y -1, respectivamente, y el resto de estados tienen una recompensa de $-0{,}04$.

MODELO DE TRANSICIÓN

Una especificación de las probabilidades resultantes para cada acción y estado posible se denomina un **modelo de transición** (o simplemente «modelo», cuando no haya lugar a dudas). Se utilizará la notación $T(s, a, s')$ para denotar la probabilidad de alcanzar el estado resultante s' si se realiza la acción a en el estado s. Se considerará que las transiciones son de tipo **Markov** en el sentido expuesto en el Capítulo 15, es decir, la probabilidad de alcanzar s' a partir de s sólo depende de s y no de la historia de los estados anteriores. Por el momento, puede considerarse que $T(s, a, s')$ es una gran tabla tridimensional que contiene probabilidades. Más adelante, en la Sección 17.5 se verá que el modelo de transición puede representarse mediante una **red Bayesiana dinámica**, vista en el Capítulo 15.

Para completar la definición del entorno de trabajo, debe especificarse la función de utilidad para el agente. Dado que el problema de decisión es secuencial, la función de utilidad dependerá de la secuencia de estados (un **histórico del entorno**) más que de un único estado. Más adelante en esta misma sección, se verá cómo pueden especificarse en general este tipo de funciones de utilidad. Por ahora, simplemente se considera que en cada estado s, el agente recibe una **recompensa** $R(s)$ que puede ser positiva o negativa, pero limitada en cualquier caso. En el ejemplo mostrado, la recompensa es $-0{,}04$ en todos los estados salvo para los estados terminales (que tienen recompensas $+1$ y -1). La utilidad de un histórico del entorno es tan sólo (y por ahora) la *suma* de las recompensas recibidas. Por ejemplo, si el agente alcanza el estado $+1$ después de 10 pasos, su utilidad total será de 0,6. La recompensa negativa de $-0{,}04$ supone un incentivo para que el agente alcance rápidamente el estado (4, 3), de modo que nuestro entorno es una generalización estocástica de los problemas de búsqueda del Capítulo 3. En otras palabras, el agente no se divierte viviendo en este entorno y, por lo tanto, quiere terminar el juego tan pronto como le sea posible.

RECOMPENSA

PROCESO DE DECISIÓN DE MARKOV

La especificación de un problema de decisión secuencial para un entorno completamente observable, con un modelo de transición de Markov y recompensas aditivas se denomina **proceso de decisión de Markov** o **PDM**. Estos procesos se definen por los tres componentes siguientes:

Estado Inicial: S_0
Modelo de Transición: $T(s, a, s')$
Función de Recompensa[1]: $R(s)$

La siguiente cuestión es determinar cuál es la forma de una solución al problema. Se ha visto que cualquier secuencia de acciones prefijada no resuelve el problema debido a que el agente puede terminar en un estado distinto del estado objetivo. Por lo tanto, cualquier solución debe especificarse de modo que el agente siempre sepa qué hacer en cada estado posible que pueda alcanzar. Se denomina **política** a una solución de este tipo. Habitualmente, se denota por π a una política y por $\pi(s)$ a la acción recomendada por la política π en el estado s. Si el agente tiene una política completa entonces no importa el resultado de una acción, el agente siempre sabe qué hacer a continuación.

POLÍTICA

[1] Algunas definiciones de los procesos de decisión de Markov permiten que la recompensa dependa también de la acción y del resultado, de forma que la función de recompensa se define como $R(s, a, s')$. Esto simplifica la descripción de algunos entornos pero no cambia significativamente el problema.

POLÍTICA ÓPTIMA

Cada vez que se ejecuta una política dada partiendo del estado inicial, la naturaleza estocástica del entorno ocasionará un histórico del entorno diferente. La calidad de la política se mide, por lo tanto, evaluando la utilidad *esperada* de los posibles históricos de entorno generados por la política. Una **política óptima** es una política que presenta la mayor utilidad esperada. Se empleará la notación π^* para denotar una política óptima. Dada π^*, el agente decide qué hacer mediante la consulta de su percepción actual, que le hace saber el estado actual s y entonces, ejecuta la acción $\pi^*(s)$. Una política representa explícitamente la función del agente y es, por lo tanto, una descripción de agente reactivo sencillo, calculado a partir de la información utilizada por un agente basado en utilidades.

Una política óptima para el entorno de la Figura 17.1 se muestra en la Figura 17.2(a). Nótese que, dado que el coste de adoptar un paso es bastante pequeño en comparación con la penalización de finalizar accidentalmente en el estado (4, 2), la política óptima para el estado (3, 1) es conservadora. La política recomienda dar un rodeo en vez de elegir el camino directo y, por tanto, arriesgarse a caer en el estado (4, 2).

El equilibro entre los riesgos asumidos y las recompensas obtenidas depende del valor de $R(s)$ para los estados no terminales. La Figura 17.2(b) muestra políticas óptimas para cuatro rangos diferentes de $R(s)$. Cuando $R(s) \leq -1{,}6284$, la vida es tan desagradable que el agente va directamente a la salida más próxima, incluso si ésta es el valor -1. Cuando está comprendida entre $-0{,}4278 \leq R(s) \leq -0{,}0850$, la vida es bastante desagradable; el agente toma la ruta más corta para alcanzar el estado $+1$ y está dispuesto a arriesgarse a caer en el estado -1 accidentalmente. En concreto, el agente toma el atajo si está en el estado (3, 1). Sólo cuando la vida es ligeramente aburrida ($-0{,}0221 < R(s) < 0$), la política óptima no toma *en absoluto ningún riesgo*. En los estados (4, 1) y (3, 2), el agente huye del estado -1 de forma que no pueda caer en él accidentalmente, incluso aunque esto signifique chocar con la pared unas cuantas veces. Finalmente, si

Figura 17.2 (a) Una política óptima para el entorno estocástico con $R(s) = -0{,}04$ para los nodos no terminales. (b) Políticas óptimas para cuatro rangos diferentes de $R(s)$.

$R(s) > 0$, la vida es tan agradable que el agente evita *ambas* salidas. En tanto en cuanto que las acciones para los estados (4, 1), (3, 2) y (4, 2) son las que se muestran, toda política es óptima y el agente obtiene una recompensa total infinita ya que nunca alcanza un estado terminal. Sorprendentemente, ello conduce a que existan otras seis políticas óptimas para varios rangos de $R(s)$. En el Ejercicio 17.7 se pide encontrarlos.

El cuidadoso balance entre riesgo y recompensa es una característica de los procesos de decisión de Markov que no aparecen en los problemas de búsqueda determinista; además, es una característica de muchos problemas de decisión del mundo real. Por esta razón, los procesos de decisión de Markov se han estudiado en diferentes ámbitos que incluyen la IA, la investigación operativa, la economía y la teoría de control. Se han propuesto multitud de algoritmos para calcular políticas óptimas. En las Secciones 17.2 y 17.3 se describirán dos de las más importantes familias de algoritmos. Sin embargo, primero se completa la revisión sobre las funciones de utilidad y las políticas para los problemas de decisión secuenciales.

Optimalidad en problemas de decisión secuenciales

En el ejemplo de PDM de la Figura 17.1, el rendimiento del agente se midió mediante la suma de las recompensas de los estados visitados. Esta elección de medida de rendimiento no es arbitraria pero tampoco es la única posible. En esta sección se examinan las posibles elecciones para evaluar el rendimiento (es decir, diferentes funciones de utilidad sobre históricos de entornos, que se escribirán como $U_h([s_0, s_1, ..., s_n])$). Esta sección recurre a las ideas del Capítulo 16 y es un tanto técnica; los puntos principales se resumen al final.

HORIZONTE FINITO

HORIZONTE INFINITO

La primera cuestión a responder es si se tiene un **horizonte finito** o un **horizonte infinito** para la toma de decisiones. Un horizonte finito significa que hay un tiempo *prefijado N*, después del cual, no importa nada (el juego ha concluido, hablando informalmente). Por lo tanto, $U_h([s_0, s_1, ..., s_{N+k}]) = U_h([s_0, s_1, ..., s_N])$ para todo $k > 0$. Por ejemplo, supóngase que el agente comienza en la posición (3,1) en el mundo de 4×3 casillas de la Figura 17.1 y que $N = 3$. Entonces, para tener una oportunidad de alcanzar el estado $+1$, el agente debe ir directamente hacia él y la acción óptima es moverse *Arriba*. Por otro lado, si $N = 100$, entonces se tiene el tiempo suficiente como para ir por la ruta segura mediante un movimiento a la *Izquierda*. *Por lo tanto, con un horizonte finito, la acción óptima para un estado dado puede cambiar a lo largo del tiempo*. Decimos que la política óptima para un horizonte finito es **no estacionaria**. Por otra parte, sin una limitación establecida de tiempo, no hay razón de comportarse de manera diferente en los mismos estados, en diferentes instantes de tiempo. Por lo tanto, la acción óptima sólo depende del estado actual y la política óptima es **estacionaria**. Las políticas para el caso de un horizonte infinito son por la tanto más sencillas que las políticas para el caso de un horizonte finito, y en este capítulo se tratará principalmente con el caso de un horizonte infinito[2]. Nótese que «ho-

POLÍTICA NO ESTACIONARIA

POLÍTICA ESTACIONARIA

[2] Esto es para entornos observables por completo. Posteriormente se verá que para entornos parcialmente observables, el caso de un horizonte infinito no es tan sencillo.

rizonte infinito» no significa necesariamente que todas las secuencias de estados son infinitas; tan sólo significa que no existe un plazo determinado y prefijado. En concreto, puede haber secuencias de estados finitas en un PDM con un horizonte infinito que contenga un estado terminal.

La siguiente cuestión que debe decidirse es cómo calcular la utilidad de las secuencias de estados. Puede abordarse el problema al estilo de la **teoría de la utilidad multiatributo** (*véase* Sección 16.4), donde cada estado s_i se ve como un atributo de la secuencia de estados $[s_0, s_1, s_2, ...]$. Para obtener una expresión sencilla en términos de los atributos, se necesita adoptar alguna ordenación relativa a la hipótesis de independencia de preferencia. La hipótesis más natural es que las preferencias del agente respecto a las secuencias de estados sean **estacionarias**. El carácter estacionario de las preferencias significa lo siguiente: si dos secuencias de estados $[s_0, s_1, s_2, ...]$ y $[s'_0, s'_1, s'_2, ...]$ comienzan desde el mismo estado (es decir, $s_0 = s'_0$) entonces las dos secuencias deberían estar ordenadas, en relación con la preferencia, del mismo modo que las secuencias $[s_1, s_2, ...]$ y $[s'_1, s'_2, ...]$. En lenguaje coloquial, esto significa que si se comenzase mañana prefiriendo un estado futuro frente a otro, entonces si se comenzase hoy, se mantendría esa misma preferencia. La preferencia estacionaria es una hipótesis con apariencia bastante inocua pero con consecuencias muy fuertes de las que se desprende que, bajo esta hipótesis, sólo hay dos formas de asignar utilidades a las secuencias:

PREFERENCIA ESTACIONARIA

RECOMPENSAS ADITIVAS

1. **Recompensas aditivas**: la utilidad de una secuencia de estados es

$$U_h([s_0, s_1, s_2, ...]) = R(s_0) + R(s_1) + R(s_2) + ...$$

El mundo de 4×3 de la Figura 17.1 utiliza recompensas aditivas. Nótese que la adición se utilizó implícitamente en el uso de las funciones de coste de rutas que se hizo en los algoritmos de búsqueda heurística (Capítulo 4).

RECOMPENSAS DEPRECIATIVAS

2. **Recompensas depreciativas**: la utilidad de una secuencia de estados es

$$U_h([s_0, s_1, s_2, ...]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...$$

FACTOR DE DESCUENTO

donde el **factor de descuento** γ es un número entre 0 y 1. El factor de descuento representa la preferencia de un agente por las recompensas actuales frente a las futuras. Cuando γ es prácticamente 0, las recompensas en un futuro lejano se consideran insignificantes. Cuando γ es 1, las recompensas depreciativas son exactamente equivalentes a las recompensas aditivas de modo que las recompensas aditivas son un caso particular de las recompensas depreciativas. El descuento se revela como un buen modelo de las preferencias animales y humanas a lo largo del tiempo. Una tasa de descuento con valor γ es equivalente a una tasa de interés del $(1/\gamma) - 1$.

Por razones que rápidamente se aclararán, se considerará el uso de recompensas depreciativas en lo que resta del capítulo, aunque en algunas ocasiones puede considerarse que $\gamma = 1$.

Bajo la elección de horizontes infinitos está latente un problema: si en el entorno no hay un estado terminal o si el agente nunca lo alcanza, entonces todos los históricos del entorno tendrán longitud infinita y las utilidades con recompensas aditivas serán, por lo

general, infinitas. Puede convenirse que un valor de utilidad de $+\infty$ es mejor que $-\infty$ pero, comparar dos secuencias, ambas con una utilidad $+\infty$, es más complicado. Existen tres soluciones, dos de las cuales ya se han visto:

1. Con recompensas depreciativas, la utilidad de una secuencia infinita es *finita*. De hecho, si las recompensas están limitadas por $R_{\text{máx}}$ y $\gamma < 1$, se tiene

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\text{máx}} = R_{\text{máx}}/(1-\gamma) \tag{17.1}$$

   empleando la expresión de la suma de los términos de una serie geométrica infinita.

2. Si el entorno presenta estados terminales *y si el agente tiene garantías de alcanzar uno de ellos eventualmente*, entonces nunca se necesitará comparar secuencias infinitas. Una política que garantiza alcanzar un estado terminal se dice que es una **política apropiada**. Con políticas apropiadas, puede considerarse que $\gamma = 1$ (es decir, recompensas aditivas). Las tres primeras políticas mostradas en la Figura 17.2(b) son apropiadas mientras que la cuarta no. Esta última política obtiene una recompensa global infinita manteniéndose lejos de los estados terminales, siempre y cuando, la recompensa para un estado no terminal sea positiva. La existencia de políticas impropias puede ocasionar que los algoritmos habituales de resolución de procesos de decisión de Markov fallen con recompensas aditivas, por lo que se tiene una buena razón de emplear recompensas depreciativas.

POLÍTICA APROPIADA

3. Otra posibilidad es comparar secuencias infinitas en función de la **recompensa media** obtenida en cada paso. Supóngase que la posición (1, 1) en el mundo de 4×3 tiene una recompensa de 0,1 mientras que los otros estados no terminales tienen una recompensa de 0,01. Entonces una política que establezca permanecer en la posición (1, 1) tendrá una recompensa media mayor que cualquiera que establezca permanecer en otro sitio. La recompensa media es un criterio útil para algunos problemas, pero el análisis de los algoritmos basados en recompensas medias superan el ámbito de este libro.

RECOMPENSA MEDIA

Resumiendo, la utilización de recompensas depreciativas presentan las menores dificultades a la hora de evaluar una secuencia de estados. El paso final es mostrar cómo elegir entre políticas, teniendo en mente que una política dada π no genera una secuencia de estados sino un rango completo de secuencias de estados, cada uno con una probabilidad concreta, determinada por el modelo de transición para el entorno. Por lo tanto, el valor de una política es la suma *esperada* de las recompensas depreciativas obtenidas, donde la esperanza se toma sobre todas las posibles secuencias de estados que pueden darse, cuando se ejecuta la política. Una política óptima π^* vendrá dada por

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right] \tag{17.2}$$

Las dos secciones siguientes describen algoritmos para encontrar políticas óptimas.

17.2 Iteración de valores

ITERACIÓN DE VALORES

En esta sección se presenta un algoritmo, denominado **iteración de valores**, para el cálculo de una política óptima. La idea básica es la de calcular la utilidad de cada *estado* y utilizar estas utilidades para seleccionar la acción óptima en cada estado.

Utilidades de los estados

La utilidad de los estados se define en función de la utilidad de las secuencias de estados. Informalmente hablando, la utilidad de un estado es la utilidad esperada de las secuencias de estados que le pueden seguir. Obviamente, las secuencias de estados dependen de la política que se ejecute, de modo que se comienza definiendo la utilidad $U^{\pi}(s)$ en relación con una política concreta π. Sea s_t el estado en el que se encuentra el agente después de ejecutar π durante t pasos (nótese que s_t es una variable aleatoria), entonces se tiene que

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right] \tag{17.3}$$

Dada esta definición, la utilidad real de un estado, que se escribe como $U(s)$, es justo $U^{\pi^*}(s)$ (es decir, la suma esperada de recompensas depreciativas si el agente ejecutase una política óptima). Nótese que $U(s)$ y $R(s)$ son cantidades muy diferentes; $R(s)$ es la recompensa «a corto plazo» por estar en el estado s, mientras que $U(s)$ es la recompensa total «a largo plazo» por proceder partiendo de s. La Figura 17.3 muestra las utilidades para el mundo de 4×3 celdas. Nótese que las utilidades son mayores para los estados más próximos al estado terminal $+1$, dado que se requieren menos pasos para alcanzar la salida.

La función de utilidad $U(s)$ permite al agente, mediante el empleo del principio de Máxima Utilidad Esperada introducido en el Capítulo 16, elegir la acción que maximice la utilidad esperada del estado resultante:

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s, a, s')U(s') \tag{17.4}$$

3	0,812	0,868	0,918	+1
2	0,762		0,660	–1
1	0,705	0,655	0,611	0,388
	1	2	3	4

Figura 17.3 Las utilidades de los estados en el mundo 4×3, calculadas para $\gamma = 1$ y una recompensa $R(s) = -0{,}04$ para los estados no terminales.

Ahora, si la utilidad de un estado es la suma de las recompensas depreciativas desde ese punto en adelante, entonces hay una relación directa entre la utilidad de un estado y la utilidad de sus vecinos: *la utilidad de un estado es la recompensa inmediata para ese estado más la utilidad depreciada y esperada del siguiente estado, considerando que el agente elige la acción óptima*. Es decir, la utilidad de un estado viene dada por

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U(s') \tag{17.5}$$

ECUACIÓN DE BELLMAN

La Ecuación (17.5) se conoce como **ecuación de Bellman**, en honor a Richard Bellman (1957). Las utilidades de los estados (definidos por la Ecuación (17.3) como la utilidad esperada de las secuencias resultantes de estados) son soluciones del conjunto de ecuaciones de Bellman. De hecho, son las *únicas* soluciones, como se verá en las dos siguientes secciones.

A continuación, se muestra una de las ecuaciones de Bellman para el mundo de 4×3. La ecuación para el estado (1, 1) es

$$\begin{aligned} U(1,1) = -0{,}04 + \gamma \max \{ \; & 0{,}8U(1,2) + 0{,}1U(2,1) + 0{,}1U(1,1), && (\textit{Arriba}) \\ & 0{,}9U(1,1) + 0{,}1U(1,2), && (\textit{Izquierda}) \\ & 0{,}9U(1,1) + 0{,}1U(2,1), && (\textit{Derecha}) \\ & 0{,}8U(2,1) + 0{,}1U(1,2) + 0{,}1U(1,1) \; \} && (\textit{Arriba}) \end{aligned}$$

Cuando se consideran los valores de la Figura 17.3, se tiene que la acción *Arriba* es la mejor.

El algoritmo de iteración de valores

La ecuación de Bellman es la base del algoritmo de iteración de valores para la resolución de procesos de decisión de Markov. Si se tienen n estados posibles entonces existen n ecuaciones de Bellman, una para cada estado. Las n ecuaciones presentan n incógnitas (las utilidades de los estados). Por lo tanto debería resolverse simultáneamente el sistema de ecuaciones para obtener las utilidades. Hay un problema añadido: las ecuaciones son *no lineales* puesto que la operación «máx» es una operación no lineal. Mientras que los sistemas de ecuaciones lineales pueden resolverse rápidamente utilizando técnicas del álgebra lineal, los sistemas de ecuaciones no lineales son más problemáticos. Una posible vía de abordar su resolución es una aproximación *iterativa*. Comenzando con unos valores iniciales cualesquiera para las utilidades, se calcula la parte derecha de la ecuación obteniéndose un nuevo valor de utilidad para el término de la izquierda de la ecuación (actualizándose de este modo las utilidades de cada estado a partir de las utilidades de sus vecinos). Se repite el proceso hasta alcanzar un estado de equilibrio. Sea $U_i(s)$ el valor de utilidad para el estado s en la i-ésima iteración. El cuerpo de la iteración, denominado **actualización de Bellman**, viene dado por la siguiente asignación:

ACTUALIZACIÓN DE BELLMAN

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U_i(s') \tag{17.6}$$

Si se aplica la actualización de Bellman indefinidamente, está garantizado alcanzar un estado de equilibrio (*véase* la subsección siguiente), en cuyo caso los valores de utili-

dad finales deberían ser las soluciones de las ecuaciones de Bellman. De hecho, son también las *únicas* soluciones, y la política asociada (obtenida empleando la Ecuación (17.4)) es óptima. El algoritmo, identificado por la etiqueta ITERACIÓN-VALORES, se muestra en la Figura 17.4.

función ITERACIÓN-VALORES (pdm, ϵ) **devuelve** una función de utilidad
  entradas: pdm, un PDM con estados S, modelo transición T, función recompensa R, descuento γ
    ϵ, error máximo permitido en la utilidad de cualquier estado
  variables locales: U, U', vectores de utilidad para los estados de S, inicialmente cero
    δ, cambio máximo en la utilidad de cualquier estado en una iteración

repetir
  $U \leftarrow U'$: $\delta \leftarrow 0$
  para cada estado s de S **hacer**
    $U'(s) \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s, a, s') U[s']$
    si $|U'[s] - U[s]| > \delta$ **entonces** $\delta \leftarrow |U'[s] - U[s]|$
hasta $\delta < \epsilon(1 - \gamma)/\gamma$
devolver U

Figura 17.4 El algoritmo de iteración de valores para el cálculo de las utilidades de los estados. La condición de terminación se obtiene a partir de la Ecuación (17.8)

Puede aplicarse la iteración de valores al mundo de 4×3 de la Figura 17.1(a). Comenzando con los valores iniciales igual a cero, las utilidades evolucionan de la forma que se muestra en la Figura 17.5(a). Nótese cómo los estados que se encuentran a diferentes distancias del estado (4,3) acumulan recompensas negativas hasta que, en un de-

Figura 17.5 (a) Gráfico que ilustra la evolución de las utilidades de los estados seleccionados, empleando la iteración de valores. (b) El número de iteraciones k necesarias para garantizar un error máximo $\epsilon = c \cdot R_{\text{máx}}$, para diferentes valores de c y en función del factor de descuento γ.

terminado punto, se encuentra un camino hacia el estado (4,3), momento en el cual, las utilidades comienzan a incrementarse. Puede pensarse que el algoritmo de iteración de valores funciona como un *propagador de información* a través del espacio de estados, mediante cambios locales.

Convergencia de la iteración de valores

Se ha dicho que la iteración de valores converge finalmente a un único conjunto de soluciones de las ecuaciones de Bellman. En esta sección se explica por qué ocurre esto. A lo largo de la argumentación, se introducirán algunas ideas matemáticas útiles y se obtendrán algunos métodos para evaluar el error en la función de utilidad devuelta cuando el algoritmo termina prematuramente; esto es útil puesto que significa que el algoritmo no tendrá que ejecutarse indefinidamente. Esta sección es bastante técnica.

CONTRACCIÓN

El concepto básico utilizado en la demostración de que el proceso iterativo converge es la noción de **contracción**. Informalmente hablando, una contracción es una función de un argumento que, cuando sucesivamente se aplica sobre dos argumentos diferentes, da como resultado otros dos valores que están «más próximos entre sí» que los valores originales, al menos en una cantidad constante. Por ejemplo, la operación «división entre dos» es una contracción ya que, después de dividir dos números cualesquiera entre dos, su diferencia se ha dividido por dos. Nótese que la función «dividir entre dos» tiene un punto fijo, el valor cero, que no sufre modificación alguna por la aplicación de la función. A partir de este ejemplo, pueden discernirse dos propiedades importantes de las contracciones:

- Una contracción tiene únicamente un punto fijo; si fueran dos los puntos fijos, éstos no podrían acercarse cuando se aplicase la función, y ésta no sería una contracción.
- Cuando la función se aplica a cualquier argumento, el valor debe acercarse a un punto fijo (ya que éste punto no se mueve) por lo que repetidas aplicaciones de una contracción siempre alcanzan el punto fijo en el límite.

Ahora, supóngase que la actualización de Bellman (Ecuación 17.6)) se ve como un operador B que se aplica simultáneamente para actualizar la utilidad de cada estado. Se denotará por U_i el vector de utilidades de todos los estados en la i-ésima iteración. Entonces la ecuación de la actualización de Bellman puede escribirse en forma vectorial como

$$U_{i+1} \leftarrow BU_i$$

NORMA DEL MÁXIMO

A continuación, se necesita un mecanismo para medir distancias entre vectores de utilidad. Se utilizará la **norma del máximo**, que mide la longitud de un vector por la longitud de su componente mayor:

$$\|U\| = \max_s |U(s)|$$

Con esta definición, la «distancia» entre dos vectores es la diferencia máxima entre dos componentes cualesquiera que ocupen la misma posición dentro del vector de utilida-

des. El resultado principal de esta sección es el siguiente: *Sea U_i y U'_i dos vectores de utilidad cualesquiera, entonces tenemos*

$$\|B\,U_i - B\,U'_i\| \leq \gamma\,\|U_i - U'_i\| \tag{17.7}$$

Es decir, la actualización de Bellman es una contracción por un factor γ sobre el espacio de vectores de utilidad. Por lo tanto, la iteración de valores siempre converge a una única solución de las ecuaciones de Bellman.

En concreto, puede reemplazarse en la Ecuación (17.7) por las utilidades *verdaderas* U, para las cuales $B\,U = U$. Entonces obtenemos la desigualdad

$$\|B\,U_i - U\| \leq \gamma\,\|U_i - U\|$$

Así que, si se considera el término como el *error* de la estimación U_i, se ve que el error se reduce al menos en un factor γ en cada iteración. Esto significa que la iteración de valores converge exponencialmente. Puede calcularse el número de iteraciones necesarias para alcanzar una cota de error ϵ específica como sigue. Primero, de la Ecuación (17.1) se tiene que las utilidades de todos los estados están acotadas por $\pm R_{\text{máx}}/(1-\gamma)$. Esto significa que el máximo error inicial $\|U_0 - U\| \leq 2R_{\text{máx}}/(1-\gamma)$. Suponiendo que N es el número de iteraciones necesarias para no superar la cota de error dada por ϵ, entonces, dado que el error se reduce en al menos una cantidad γ en cada iteración, se necesita que $\gamma^N \cdot 2R_{\text{máx}}/(1-\gamma) \leq \epsilon$. Tomando logaritmos, se tiene que

$$N = [\log(2R_{\text{máx}}/\epsilon(1-\gamma))/\log(1/\gamma)]$$

es el número de iteraciones suficientes. La Figura 17.5(b) muestra como varía N en relación con γ, para diferentes valores de la proporción $\epsilon/R_{\text{máx}}$. La buena noticia es que, debido a la rápida convergencia, N no depende excesivamente de la proporción $\epsilon/R_{\text{máx}}$. La mala noticia es que N crece rápidamente conforme γ se aproxima a 1. Puede lograrse una convergencia más rápida si se hace que γ sea pequeño, lo que se consigue realmente es que el agente tenga un horizonte más corto, por lo que pueden obviarse los efectos de las acciones del agente a largo plazo.

La cota de error establecida en el párrafo precedente da una ida de los factores que influyen en el tiempo de ejecución del algoritmo, pero es un criterio de parada conservador algunas veces. Para obtener una condición de parada más flexible, puede emplearse el límite que relaciona el error con el tamaño de la actualización de Bellman, para una iteración cualquiera. A partir de la propiedad de contracción (Ecuación (17.7)), puede demostrarse que, si la modificación es pequeña (es decir, no cambia mucho la utilidad de un estado), entonces el error, en comparación con la función de utilidad verdadera, es también pequeño. De forma más precisa,

$$\text{si } \|U_{i+1} - U_i\| < \epsilon(1-\gamma)/\gamma \quad \text{entonces} \quad \|U_{i+1} - U\| < \epsilon \tag{17.8}$$

Esta es la condición de terminación utilizada por el algoritmo ITERACIÓN-VALORES mostrado en la Figura 17.4.

Hasta ahora, se ha analizado el error de la función de utilidad devuelta por el algoritmo de iteración de valores. *Sin embargo, de lo que realmente debe preocuparse el agente es de lo bien que actuará si toma sus decisiones con base en esta función de utilidad.*

Supóngase que después de i iteraciones del algoritmo, el agente tiene una estimación U_i para la utilidad verdadera U y obtiene la política MUE π_i mediante la anticipación de un paso, en función de U_i (como en la Ecuación (17.4)). ¿El comportamiento resultante será tan bueno como el comportamiento óptimo? Esta es una pregunta crucial para cualquier agente real y resulta que la respuesta es que sí. Si $U^{\pi_i}(s)$ es la utilidad obtenida si se ejecuta π_i partiendo del estado s, y se define la **pérdida de la política**, $\|U^{\pi_i} - U\|$, como la mayor cantidad que el agente puede perder por la ejecución de π_i en vez de la política óptima π^*. La pérdida de la política π_i está relacionada con el error en U_i mediante la siguiente desigualdad:

PÉRDIDA DE LA POLÍTICA

$$\text{si} \quad \|U_i - U\| < \epsilon \quad \text{entonces} \quad \|U^{\pi_i} - U\| < 2\epsilon\gamma/(1-\gamma) \tag{17.9}$$

En la práctica, a menudo se tiene que π_i se convierte en la política óptima bastante antes de que U_i converja. La Figura 17.6 muestra cómo el error máximo en U_i y el error en la pérdida de la política se aproximan a cero, conforme avanza el proceso iterativo, para el entorno de 4×3 celdas con $\gamma = 0{,}9$. La política π_i es óptima cuando $i = 4$, aunque el error máximo de U_i es aún de 0,46.

Figura 17.6 El error máximo $\|U_i - U\|$ en las estimaciones de los valores de utilidad y la pérdida de la política $\|U^{\pi_i} - U\|$ comparada con la política óptima, en función de las iteraciones dadas por el algoritmo de iteración de valores

En este punto, se dispone de todo lo necesario para utilizar la iteración de valores en la práctica. Se sabe que converge a los valores de utilidad correctos, puede limitarse el error en la estimación de la utilidad si el algoritmo se para después de un número finito de iteraciones y puede limitarse la pérdida de la política resultante de ejecutar la política derivada del principio de máxima utilidad esperada. Un comentario final se refiere a que todos los resultados de esta sección están condicionados a que la tasa de descuento sea menor que 1 ($\gamma < 1$). Si $\gamma = 1$ y en el entorno se tienen estados terminales, entonces puede derivarse un conjunto similar de resultados sobre la convergencia y los límites del error, siempre que se satisfagan determinadas condiciones técnicas.

17.3 Iteración de políticas

En la sección anterior, se observó que es posible alcanzar una política óptima, incluso cuando la estimación de la función de utilidad es imprecisa. Si una acción es claramente mejor que todas las restantes entonces las magnitudes exactas de las utilidades de los estados involucrados no necesitan ser precisas. Esta visión sugiere una vía alternativa a la hora de encontrar políticas óptimas. El algoritmo de **iteración de políticas** alterna los dos pasos siguientes, partiendo de alguna política inicial π_0:

ITERACIÓN DE POLÍTICAS

EVALUACIÓN DE LA POLÍTICA

MEJORA DE LA POLÍTICA

- **Evaluación de la política:** dada una política π_i, calcular $U_i = U^{\pi_i}$, la utilidad de cada estado si π_i fuese ejecutada.
- **Mejora de la política:** calcular una nueva política π_{i+1} con base en la aplicación del principio de MUE, empleando la anticipación de un paso en función de U_i (como en la Ecuación (17.4)).

El algoritmo concluye cuando la etapa de mejora del algoritmo no produce un cambio en las utilidades. En este punto, se sabe que la función de utilidad U_i es un punto fijo de la actualización de Bellman, por lo que es una solución de las ecuaciones de Bellman, y que π_i debe ser una política óptima. Dado que para un espacio finito de estados hay un conjunto finito de políticas y que, en cada iteración, puede demostrarse que se mejora la política, necesariamente, la iteración de políticas debe concluir. El algoritmo se muestra en la Figura 17.7.

```
función ITERACIÓN-POLÍTICAS (pdm) devuelve una política
  entradas: pdm, un PDM con estados S, modelo transición T
  variables locales: U, U′, vectores de utilidad para los estados S, inicialmente cero
                     π, un vector de políticas indexado por estado, inicialmente al azar

  repetir
    U ← EVALUACIÓN-POLÍTICA(π, U, pdm)
    sin_cambios? ← verdadero
    para cada estado s de S hacer
      si máx_a Σ_s′ T(s, a, s′)U[s′] > Σ_s′ T(s, π[s], s′)U[s′] entonces
        π[s] argmax_a Σ_s′ T(s, a, s′)U[s′]
        sin_cambios? ← falso
  hasta sin_cambios?
  devolver P
```

Figura 17.7 El algoritmo de iteración de políticas para el cálculo de una política óptima.

La etapa de mejora de la política es obviamente sencilla pero, ¿cómo implementar la rutina EVALUACIÓN-POLÍTICA? Pues bien, resulta que hacerlo resulta mucho más simple que resolver las ecuaciones de Bellman normales (que es lo que la iteración de valores realiza), ya que la acción en cada estado está fijada por la política. En la *i*-ésima

iteración, la política π_i determina la acción $\pi_i(s)$ en el estado s. Esto significa que se tiene una versión simplificada de la ecuación de Bellman (17.5) que relaciona la utilidad del estado s (bajo la política π_i) con las utilidades de sus vecinos:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \tag{17.10}$$

Por ejemplo, supóngase que π_i es la política mostrada en la Figura 17.2(a). Entonces se tiene que $\pi_i(1, 1) = Arriba$, $\pi_i(1, 2) = Arriba$, y así sucesivamente. Las ecuaciones de Bellman simplificadas son

$$\begin{aligned} U_i(1, 1) &= 0{,}8\, U_i(1, 2) + 0{,}1\, U_i(1, 1) + 0{,}1\, U_i(2, 1) \\ U_i(1, 2) &= 0{,}8\, U_i(1, 3) + 0{,}2\, U_i(1, 2) \\ &\vdots \end{aligned}$$

La cuestión importante es que estas ecuaciones son *lineales*, puesto que se ha eliminado de ellas el operador «máx». Para n estados, se tienen n ecuaciones lineales con n incógnitas, que pueden resolverse exactamente en un tiempo del orden de $O(n^3)$ mediante métodos normales del álgebra lineal.

Para espacios de estados pequeños, la evaluación de la política empleando los métodos de solución exacta son a menudo la aproximación más eficiente. Para grandes espacios de estados, un tiempo de ejecución del orden de $O(n^3)$ puede ser prohibitivo. Afortunadamente, no es necesario evaluar exactamente la política. En su lugar, puede realizarse un número limitado de pasos simplificados de la iteración de valores (simplificado porque la política está fijada) para obtener una aproximación razonablemente buena de las utilidades. La actualización de Bellman simplificada para este proceso viene dada por la siguiente asignación:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

ITERACIÓN DE POLÍTICAS MODIFICADA

y se repite k veces para producir la siguiente estimación de los valores de utilidad. El algoritmo resultante se denomina **iteración de políticas modificada**. A menudo, es mucho más eficiente que la iteración de políticas estándar o que la iteración de valores.

ITERACIÓN DE POLÍTICAS ASÍNCRONA

El algoritmo descrito hasta ahora necesita actualizar el valor de utilidad o la política para todos los estados al mismo tiempo. Esto no es estrictamente necesario. De hecho, en cada iteración, puede elegirse *cualquier subconjunto* de estados y aplicar *cualquier* tipo de actualización (mejora de la política o iteración de valores simplificada) a este subconjunto. Esta versión más general del algoritmo se denomina **iteración de políticas asíncrona**. Bajo ciertas condiciones sobre la política inicial y la función de utilidad, está garantizado que la iteración de políticas asíncrona converge hacia la política óptima. La libertad de elegir cualquier subconjunto de estados con los que trabajar significa que pueden diseñarse algoritmos heurísticos mucho más eficientes (por ejemplo, algoritmos que se concentren en actualizar los valores de los estados que son más probables de alcanzarse por una buena política). Esto tiene mucho sentido en la vida real: si uno no tiene intención de arrojarse por un precipicio, uno no debería perder tiempo en preocuparse por el valor exacto de la utilidad de los estados resultantes.

17.4 Procesos de decisión de Markov parcialmente observables

La descripción de los procesos de decisión de Markov hecha en la Sección 17.1 asume que el entorno era **completamente observable**. Bajo esta hipótesis, el agente siempre sabe el estado en el que se encuentra. Esto, combinado con la hipótesis de Markov en cuanto al modelo de transición, se traduce en que la política óptima sólo depende del estado actual. Cuando el entorno sólo es **observable parcialmente**, uno podría decir que la situación es mucho menos clara. El agente no conoce necesariamente en qué estado se encuentra, por lo que no puede ejecutar la acción $\pi(s)$ recomendada para ese estado. Además, la utilidad de un estado s y la acción óptima en el estado s no sólo dependen de s, sino que también dependen de lo *que el agente conozca* cuando se encuentre en el estado s. Por estas razones, los **procesos de decisión de Markov parcialmente observables** (PDMPO) son habitualmente mucho más complicados que los procesos de decisión de Markov normales. Por otro lado, este tipo de procesos no se pueden obviar ya que el mundo real es un ejemplo de ellos.

PDM OBSERVABLE PARCIALMENTE

Como ejemplo, considérese otra vez el mundo de 4×3 celdas descrito en la Figura 17.1, pero ahora supóngase que el agente *no tiene sensores de ninguna clase* y que *no tiene ni idea de dónde se encuentra*. De forma más precisa, supóngase que cualquiera de los nueve estados no terminales tienen la misma probabilidad de ser el estado inicial (Figura 17.8(a)). Claramente, si el agente *conociese* que está en la posición (3,3), se movería a la *Derecha*; si el agente *supiese* que está en la posición (1,1), se movería *Arriba*. Ahora bien, como puede estar en cualquier posición, ¿qué haría el agente? Una respuesta posible es que el agente, en un primer momento, actuaría con el fin de reducir su incertidumbre y sólo entonces trataría de dirigirse hacia la salida $+1$. Por ejemplo, si el agente se mueve a la *Izquierda* cinco veces entonces es bastante probable que se encuentre junto a la pared izquierda (Figura 17.8(b)). Si después se mueve *Arriba* otras cinco veces es probable que se encuentre en la parte superior, probablemente, en la esquina superior izquierda (Figura 17.8(c)). Por último, si se mueve cinco veces a la *Derecha* tiene una probabilidad considerable (alrededor de un 77,5 por ciento) de alcanzar el estado terminal $+1$ (Figura 17.8(d)). Si a

Figura 17.8 (a) Distribución de probabilidad inicial acerca de la localización del agente. (b) Después de moverse cinco veces a la *Izquierda*. (c) Después de moverse cinco veces *Arriba*. (d) Después de moverse cinco veces a la *Derecha*.

partir de entonces continúa moviéndose a la derecha, incrementará sus posibilidades hasta un 81,8 por ciento. Por lo tanto, esta política es sorprendentemente segura, pero si la adoptase, el agente sería bastante lento en alcanzar la salida y tendría una utilidad esperada de tan sólo 0,08, aproximadamente. La política óptima, que se describirá en breve, lo hace mucho mejor.

Para tratar con procesos de decisión de Markov parcialmente observables, primero deben definirse de forma apropiada. Un PDMPO tiene los mismos elementos que un PDM (el modelo de transición $T(s, a, s')$ y la función de recompensa $R(s)$), pero tiene, además, un **modelo de observación** $O(s, o)$ que determina la probabilidad de percibir la observación o en el estado s[3]. Por ejemplo, un agente sin sensores sólo dispone de una posible observación (la observación vacía) y ésta ocurre con probabilidad 1 en cada estado.

MODELO DE OBSERVACIÓN

En los Capítulos 3 y 12, cuando se estudiaron los problemas de planificación no deterministas y observables parcialmente, se identificó el **estado de creencia** (el conjunto de estados actuales en los que el agente podría estar) como un concepto clave para describir y calcular las soluciones. Con los procesos de decisión de Markov parcialmente observables, se modifica un poco este concepto. Un estado de creencia b es ahora una *distribución de probabilidad* sobre todos los posibles estados. Por ejemplo, el estado de creencia inicial de la Figura 17.8(a) puede escribirse como $\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \rangle$. Mediante $b(s)$ se denotará la probabilidad asignada al estado actual s por el estado de creencia b. El agente puede calcular su estado de creencia actual como la distribución de probabilidad condicionada sobre los estados actuales, conocida la secuencia de observaciones y acciones realizadas hasta ahora. Esencialmente, ésta es la tarea de **filtrado** (*véase* el Capítulo 15). La ecuación recursiva básica de filtrado (Ecuación (15.3)) muestra cómo calcular el nuevo estado de creencia a partir del estado de creencia anterior y la nueva observación. Para el caso de un PDMPO, debe considerarse también una acción y una notación ligeramente diferente, pero fundamentalmente, el resultado es el mismo. Si $b(s)$ era el estado de creencia previo y el agente realiza la acción a y percibe la observación o, entonces el nuevo estado de creencia viene dado por

$$b'(s') = \alpha\, O(s', o) \sum_s T(s, a, s')\, b(s) \tag{17.11}$$

donde α es una constante de normalización que garantiza que la suma de los componentes del estado de creencia sea 1. Puede abreviarse esta ecuación introduciendo el operador SIGUIENTE y rescribiéndola como $b' = \text{SIGUIENTE}(b, a, o)$.

La idea fundamental, necesaria para entender los procesos de decisión de Markov parcialmente observables es que: *la acción óptima depende exclusivamente del estado de creencia actual del agente*. Es decir, la política óptima puede describirse mediante una correspondencia $\pi^*(b)$ que asocie estados de creencias con acciones. Por lo tanto, *no* depende del estado *actual* en el que se encuentre el agente. Esto es una buena cosa ya que el agente no sabe cuál es su estado actual y todo lo que conoce es

[3] Esencialmente, el modelo de observación es idéntico al **modelo sensitivo** de los procesos temporales, ya descrito en el Capítulo 15. Como con las funciones de recompensa en los procesos de decisión de Markov, el modelo de observación puede depender también de la acción y del estado resultante, pero este cambio no es fundamental.

su estado de creencia. Por lo tanto, el ciclo de decisión de un agente PDMPO es el siguiente:

1. Dado el estado de creencia actual b, ejecutar la acción $a = \pi^*(b)$.
2. Recibir la observación o.
3. Establecer el estado de creencia actual con el valor devuelto por SIGUIENTE(b, a, o) y volver al paso 1.

Ahora puede plantearse que los procesos parcialmente observables necesitan realizar una búsqueda en el espacio de estados de creencia del mismo modo que los métodos vistos en el Capítulo 3, para los problemas de contingencia y con ausencia de percepción. La diferencia principal es que el espacio de estados de creencia de un PDMPO es *continuo* ya que su estado de creencia es una distribución de probabilidad. Por ejemplo, un estado de creencia para el mundo de 4×3 es un punto en el espacio continuo n-dimensional, con $n = 11$. Una acción cambia el estado de creencia, no sólo el estado físico, de forma que éste se evalúa de acuerdo con la información que el agente adquiere como un resultado. Por lo tanto, los procesos parcialmente observables incluyen el valor de la información (Sección 16.6) como un componente del problema de decisión.

A continuación se examinan, con más detalle, los resultados de las acciones. En concreto, se calcula la probabilidad de que un agente, encontrándose en el estado de creencia b, alcance el estado de creencia b' después de ejecutar la acción a. Ahora, si se conociese la acción *y la observación subsiguiente*, la Ecuación (17.11) proporcionaría una actualización *determinista* del estado de creencia: $b' = \text{SIGUIENTE}(b, a, o)$. Por supuesto, la observación subsiguiente no se conoce aún, por lo que el agente podría terminar en uno de entre los varios estados de creencia posibles b', dependiendo de la observación que ocurra. La probabilidad de percibir la observación o, conociendo que se realizó la acción a partiendo del estado de creencia b, viene dada por la suma sobre todos los nuevos estados s' que el agente puede alcanzar:

$$\begin{aligned} P(o|a, b) &= \sum_{s'} P(o|a, s', b)P(s'|a, b) \\ &= \sum_{s'} O(s', o)P(s'|a, b) \\ &= \sum_{s'} O(s', o) \sum_{s} T(s, a, s')b(s) \end{aligned}$$

Si se denota por $\tau(b, a, b')$ a la probabilidad de alcanzar b' partiendo de b, dada la acción a, entonces se tiene que:

$$\begin{aligned} \tau(b, a, b') = P(b'|a, b) &= \sum_{o} P(b'|o, a, b)P(o|a, b) \\ &= \sum_{o} P(b'|o, a, b) \sum_{s'} O(s', o) \sum_{s} T(s, a, s')b(s) \end{aligned} \tag{17.12}$$

donde $P(b'|o, a, b)$ es 1 si $b' = \text{SIGUIENTE}(b, a, o)$ y 0 en otro caso.

La Ecuación (17.12) puede verse como la definición del modelo de transición para el espacio de estados de creencia. Puede definirse también una función de recompensa

para los estados de creencia (es decir, la recompensa esperada para los estados actuales en los que el agente puede encontrarse):

$$\rho(b) = \sum_s b(s)R(s).$$

También parece que $\tau(b, a, b')$ y que $\rho(b)$ definen conjuntamente un PDM *observable* sobre el espacio de estados de creencia. Además, puede demostrarse que la política óptima para este PDM, $\pi^*(b)$, es también una política óptima para el problema de decisión original, parcialmente observable. En otras palabras, *la resolución de un PDMPO sobre un espacio de estados físicos puede reducirse a resolver un PDM sobre el correspondiente espacio de estados de creencia*. Este hecho es quizá menos sorprendente, si se recuerda que el espacio de estados de creencia siempre es observable para el agente, por definición.

Nótese que, aunque hemos reducido los procesos parcialmente observables a un PDM convencional, el PDM resultante tiene un espacio de estados continuo (y por lo general, con una dimensionalidad mayor). Ninguno de los algoritmos para tratar procesos de decisión de Markov descritos en las Secciones 17.2 y 17.3 son aplicables directamente a los procesos de decisión de Markov resultantes de la transformación. Parece que *pueden* desarrollarse versiones de los algoritmos de iteración de valores y de políticas para aplicarlas a los problemas de decisión de Markov con estados continuos. La idea básica es que una política $\pi(b)$ puede representarse como un conjunto de *regiones* del espacio de estados de creencia, cada una de las cuales está asociada con una acción óptima concreta[4]. La función de valor asocia una función *lineal* distinta de b a cada región. Cada paso de la iteración de valores (o de políticas) perfila los límites de las regiones y puede introducir nuevas regiones.

Los detalles del algoritmo se escapan del ámbito de este libro, pero se proporciona a continuación la solución para el mundo 4×3 sin percepción. La política óptima es la siguiente:

[*Izquierda*, *Arriba*, *Arriba*, *Derecha*, *Arriba*, *Arriba*, *Derecha*, *Arriba*, *Arriba*, *Derecha*, *Arriba*, *Derecha*, *Arriba*, *Derecha*, *Arriba*, ...].

La política es una secuencia porque el problema es *determinista* en el espacio de estados de creencias (no hay observaciones). El «truco» que encierra esta solución es hacer que el agente se mueva a la *Izquierda* una vez para asegurar que *no* está en la posición (4, 1), de modo que esté bastante seguro de mantener el movimiento *Arriba* y a la *Derecha* para alcanzar la salida +1. El agente alcanza la salida +1 un 86,6 por ciento de las veces y lo hace de forma mucho más rápida que la política indicada al comienzo de la sección, por lo que su utilidad esperada es de 0,38 frente al valor de 0,08 obtenida por la primera.

Para PDMPOs mucho más complejos con observaciones, encontrar aproximaciones a las políticas óptimas es muy difícil (pertenecen a la categoría de problemas NP-duros). Los problemas con apenas una docena de estados son a menudo intratables. La siguiente sección describe un método de aproximación diferente para la resolución de PDMPOs, basado en una búsqueda con anticipación.

---

[4] Para algunos problemas de decisión de Markov parcialmente observables, la política óptima tiene un elevadísimo número de regiones, por lo que la sencilla aproximación de la lista de regiones fracasa y son necesarios métodos más ingeniosos para encontrar, incluso, una aproximación.

17.5 Agentes basados en la teoría de la decisión

En esta sección, se esboza una aproximación comprensiva al diseño de agentes en entornos estocásticos y observables parcialmente. Los elementos básicos ya son familiares:

- Los modelos de transición y observación se representan mediante una **red Bayesiana dinámica** (ya descrita en el Capítulo 15).
- La red Bayesiana dinámica se amplía con la inclusión de nodos de decisión y de utilidad, al estilo de las redes de decisión descritas en el Capítulo 16. El modelo resultante se denomina **red de decisión dinámica**, abreviadamente RDD.
- Un algoritmo de filtrado se utiliza para incorporar cada percepción y acción nuevas y para actualizar la representación del estado de creencia.
- La decisión se realiza mediante la proyección hacia delante de todas las posibles secuencias de acciones y la elección de la mejor.

RED DE DECISIÓN DINÁMICA

La principal ventaja de utilizar una red Bayesiana dinámica para representar los modelos de transición y de percepción es que con ello se descompone la descripción del estado en un conjunto de variables aleatorias, del mismo modo que los algoritmos de planificación utilizan representaciones lógicas para descomponer el espacio de estados explorado por los algoritmos de búsqueda. Por lo tanto el diseño del agente es una implementación práctica del **agente basado en utilidades**, esbozado en el Capítulo 2.

Dado que se emplean redes Bayesianas dinámicas, se retoma la notación del Capítulo 15, donde $\mathbf{X}_t$ se refiere al conjunto de variables de estado en el instante t y $\mathbf{E}_t$ se refiere a las variables de evidencia. Por lo tanto, donde hasta ahora se ha empleado s_t (el estado en el instante t) en este capítulo, ahora se utilizará $\mathbf{X}_t$. Se utilizará A_t para referirse a la acción realizada en el instante t, por lo que el modelo de transición $T(s, a, s')$ es equivalente a escribir $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, A_t)$ y el modelo de observación $O(s, o)$ es equivalente a $P(\mathbf{E}_t|\mathbf{X}_t)$. Se utilizará R_t para referirse a la recompensa recibida en el instante de tiempo t y U_t para referirse a la utilidad del estado en el instante t. Con esta notación, una red de decisión dinámica tiene un aspecto parecido al mostrado en la Figura 17.9.

Las redes de decisión dinámica proporcionan una representación precisa de grandes procesos de decisión de Markov parcialmente observables, por lo que pueden utilizarse como datos de entrada para cualquiera de los algoritmos que tratan con este tipo de proceso, incluyendo los métodos de iteración de valores y de políticas. Esta sección se centra en los métodos con anticipación que proyectan secuencias de acciones hacia el futuro, a partir del estado de creencia actual, de la misma forma que los algoritmos sobre juegos del Capítulo 6. En la red de la Figura 17.9 se han proyectado tres pasos hacia el futuro; la decisión actual y las futuras, así como las observaciones y recompensas futuras se desconocen. Nótese que en la red se incluyen nodos para representar las *recompensas* para los estados $\mathbf{X}_{t+1}$ y $\mathbf{X}_{t+2}$, pero para el estado $\mathbf{X}_{t+3}$ se representa su *utilidad*. Esto es así puesto que el agente debe maximizar la suma (depreciada) de todas la futuras recompensas y $U(\mathbf{X}_{t+3})$ representa, precisamente, la re-

Figura 17.9 Estructura genérica de una red de decisión dinámica. Las variables con un valor conocido aparecen sombreadas. El instante de tiempo actual es t y el agente debe decidir qué hacer (es decir, elegir un valor para A_t). La red se ha proyectado hacia un futuro tres pasos y se representan las recompensas futuras, así como la utilidad del estado en el horizonte adelantado.

compensa para el estado $\mathbf{X}_{t+3}$ y la de todos los estados sucesivos. Al igual que en el Capítulo 6, se considera que se dispone de U sólo de forma aproximada: si los valores de utilidad exactos estuvieran disponibles, no sería necesario adelantarse más allá de un paso en el tiempo.

La Figura 17.10 muestra parte del árbol de búsqueda correspondiente a la RDD con tres pasos de anticipación mostrada en la Figura 17.9. Cada nodo triangular representa el estado de creencia en el que se encuentra el agente cuando toma una decisión A_{t+i} ($i = 0, 1, 2, ...$). Los nodos circulares corresponden a elecciones por parte del entorno, es decir, qué observación $\mathbf{E}_{t+i}$ ocurre. Nótese que no hay nodos aleatorios asociados a los resultados de la acción; esto es así porque la actualización del estado de creencia para una acción es determinista, sin tener en cuenta el resultado actual.

Figura 17.10 Parte de la solución adelantada de la RDD mostrada en la Figura 17.9.

El estado de creencia en cada nodo triangular puede calcularse mediante la aplicación del algoritmo de filtrado a la secuencia de observaciones y acciones que conducen a él. De esta forma, el algoritmo considera el hecho de que, para la decisión A_{t+i}, el agente *tendrá* disponibles las observaciones $\mathbf{E}_{t+1}$, ..., $\mathbf{E}_{t+i}$, incluso aunque no conozca cuáles serán esas observaciones en el instante de tiempo t. De esta forma, un agente basado en la teoría de la decisión considerará automáticamente el valor de la información y ejecutará acciones de adquisición de información cuando sea oportuno.

A partir del árbol de búsqueda puede obtenerse una decisión sin más recuperando los valores de utilidad de las hojas, calculando un promedio en los nodos aleatorios y considerando el máximo de los nodos de decisión. Este proceso es similar al desempeñado por el algoritmo EXPECTIMINIMAX para árboles de juego con nodos aleatorios, con las excepciones de que (1) también puede haber recompensas para los estados que no son hojas y (2) los nodos de decisión se corresponden con estados de creencia, más que con estados reales. La complejidad temporal de una búsqueda exhaustiva de profundidad d es del orden de $O(|D|^d \cdot |\mathbf{E}|^d)$, donde $|D|$ es el número de acciones disponibles y $|\mathbf{E}|$ es el número de posibles observaciones. Para problemas en los que el factor de descuento γ no está muy cerca de 1, una búsqueda poco profunda es a menudo suficientemente buena para obtener decisiones cuasi-óptimas. También es posible simplificar el paso de promediar los nodos aleatorios mediante el muestreo del conjunto de posibles observaciones en vez de promediar sobre la totalidad de posibles observaciones. Existen otras vías diferentes para obtener rápidamente soluciones aproximadas buenas, pero se aplaza su explicación al Capítulo 21.

Los agentes basados en la teoría de la decisión que utilizan redes de decisión dinámicas presentan una serie de ventajas sobre otros agentes más sencillos, cuyo diseño se ha presentado en los capítulos anteriores. En concreto, pueden manejar entornos inciertos y parcialmente observables y pueden fácilmente revisar su «planes» para tratar con observaciones inesperadas. Con los modelos sensitivos apropiados, pueden tratar con fallos en los sensores y planificar la recopilación de información. Exhiben una «degradación aceptable» ante la presencia de fuertes restricciones temporales y entornos complejos, gracias a la utilización de aproximaciones. ¿Y cuál es el inconveniente? El defecto más importante del algoritmo basado en redes de decisión dinámicas es su dependencia de una búsqueda anticipada, al igual que los algoritmos de búsqueda en el espacio de estados de la Parte II. En la Parte IV se explicará cómo la capacidad de considerar planes abstractos, parcialmente ordenados mediante una búsqueda guiada por objetivos, proporciona un incremento significativo del poder de resolución de problemas, especialmente reseñable cuando se combina con bibliotecas de planes. Se ha intentado extender estos métodos al dominio probabilista, pero hasta el momento, todos los intentos se han mostrado ineficientes. Un segundo problema es la naturaleza básicamente proposicional del lenguaje de las redes de decisión dinámicas. Sería muy conveniente tener la capacidad de aplicar algunas de las ideas de los lenguajes probabilistas de primer orden de la Sección 14.6 al problema de la toma de decisiones. Los resultados de la investigación actual han puesto de manifiesto que esta extensión es posible y que tiene beneficios significativos, tal y como se pone de manifiesto en las notas finales de este capítulo.

17.6 Decisiones con varios agentes: teoría de juegos

Este capítulo se ha centrado en la toma de decisiones en entornos con incertidumbre. Pero, ¿qué ocurre si la incertidumbre se debe a la presencia de otros agentes y a las decisiones que puedan adoptar? Y, ¿si las decisiones de esos agentes resultan estar influenciadas por nuestras decisiones? Con anterioridad ya se trató esta cuestión, cuando se estudiaron los juegos en el Capítulo 6. Sin embargo, allí se trataba de juegos donde los jugadores se turnan y tienen información perfecta, por lo que la búsqueda minimax podía usarse para encontrar movimientos óptimos. En esta sección se estudian los aspectos de la **teoría de juegos** que pueden utilizarse para analizar juegos con movimientos simultáneos. Para simplificar las cosas, se examinarán primero los juegos que sólo son de un movimiento. La palabra «juego» y la simplificación a un único movimiento puede hacer que este asunto parezca trivial, pero de hecho, la teoría de juegos se utiliza en situaciones de toma de decisiones muy serias como la adopción de contramedidas en situaciones de quiebra, la subasta del espectro radioeléctrico, las decisiones de fijación de precios y desarrollo de productos y la defensa nacional, situaciones en las que se manejan grandes cantidades de dinero y están en juego vidas humanas. La teoría de juegos puede emplearse al menos de dos formas:

TEORÍA DE JUEGOS

1. **Diseño de Agentes:** la teoría de juegos puede analizar las decisiones del agente y calcular la utilidad esperada de cada decisión (bajo la hipótesis de que otros agentes están actuando de forma óptima de acuerdo con la teoría de juegos). Por ejemplo, en el juego denominado «**pares o nones**» **con dos dedos**, dos jugadores, I y P, muestran a la vez uno o dos dedos. Sea f el número total de dedos que muestran los jugadores. Si f es impar entonces I gana los f dólares (que tiene que darle P), mientras que si f es par, P gana los f dólares (que le tiene que dar en este caso I). La teoría de juegos puede determinar la mejor estrategia a utilizar frente a un jugador racional y calcular el beneficio esperado para cada jugador[5].
2. **Diseño de Mecanismos:** cuando un entorno está cohabitado por muchos agentes, puede ser posible definir las reglas del entorno (es decir, el juego que los agentes deben jugar) de forma que el bien común de todos los agentes se maximice cuando cada agente adopte la solución que, desde el punto de vista de la teoría de juegos, maximiza su propia utilidad. Por ejemplo, la teoría de juegos puede ayudar a diseñar protocolos para un conjunto de rutas de tráfico de Internet de forma que cada ruta tiene un incentivo para actuar de forma que se maximice el rendimiento global. El diseño de mecanismos puede emplearse también para construir sistemas multiagentes inteligentes que resuelvan problemas complejos de forma distribuida, sin necesidad de que cada agente sepa qué parte del problema global está resolviendo.

[5] «Pares o nones» es una versión recreativa de un **juego de inspección**. Es estos juegos, un inspector elige un día para inspeccionar un servicio (como un restaurante o una planta de armas biológicas) y el operador del servicio elige un día para ocultar todas las deficiencias. El inspector gana si los días elegidos son diferentes y el operador gana cuando son los mismos.

Un juego en la teoría de juegos está definido por los componentes siguientes:

JUGADORES

- **Jugadores** o agentes que serán quienes tomen las decisiones. Los juegos con dos jugadores han recibido una mayor atención aunque los juegos de n jugadores (con $n > 2$) también son habituales. Se emplearán nombres con la inicial en mayúsculas para designar a los jugadores, como *Alicia* y *Roberto* o I y P.

ACCIONES

- **Acciones** que los jugadores pueden elegir. Se emplearán nombres en minúsculas para referirnos a ellas, como por ejemplo *una* o *testificar*. Los jugadores pueden o no tener el mismo conjunto de acciones disponibles.

MATRIZ DE BALANCES FINALES

- Una **matriz de balances finales** que proporciona la utilidad de cada jugador para cada combinación de acciones de todos los jugadores. La matriz de balances finales para el juego «pares o nones» es la siguiente:

	I: *uno*	I: *dos*
P: *uno*	$P = 2,\ I = -2$	$P = -3,\ I = 3$
P: *dos*	$P = -3,\ I = 3$	$P = 4,\ I = -4$

Por ejemplo, en la celda de la esquina inferior derecha se muestra que cuando I elige la acción *dos* y P también elige *dos*, el balance final es de 4 para P y de -4 para I.

ESTRATEGIA PURA

ESTRATEGIA MIXTA

PERFIL DE ESTRATEGIA

RESULTADO

SOLUCIÓN

Cada jugador debe adoptar y, después, ejecutar una **estrategia** (que es el nombre utilizado en la teoría de juegos para una política). Una **estrategia pura** es una política determinista que indica la acción concreta a realizar en cada situación. Para un juego de un movimiento, una estrategia pura es sólo una única acción. El análisis de los juegos conduce a la idea de **estrategia mixta**, que es una política aleatoria que selecciona acciones concretas de acuerdo con la distribución de probabilidad concreta de cada una de las acciones. La estrategia mixta que seleccione la acción a con probabilidad p y la acción b en otro caso se escribe como $[p: a; (1 - p): b]$. Por ejemplo, una estrategia mixta para el juego «pares o nones» con dos dedos puede ser [0,5: *uno*; 0,5: *dos*]. Un **perfil de estrategia** es una asignación de una estrategia mixta para cada jugador. Dado un perfil de estrategia, el **resultado** del juego es un valor numérico para cada jugador.

Una **solución** del juego es un perfil de estrategia en la que cada jugador adopta una estrategia racional. Se verá que la cuestión más importante de la teoría de juegos es definir qué significa «racional» cuando cada agente elige sólo parte del perfil de estrategia que determina el resultado. Es importante darse cuenta de que los resultados son realmente los resultados del juego, mientras que las soluciones son construcciones teóricas utilizadas para analizar el juego. Se verá que sólo algunos juegos tienen una única solución en el espacio de estrategias mixtas. Esto no significa que el jugador deba, literalmente, adoptar una estrategia mixta para comportarse racionalmente.

Considérese el siguiente relato: dos presuntos ladrones, *Alicia* y *Roberto*, son detenidos *in fraganti* cerca del lugar de un robo y la policía les interroga por separado. Ambos saben que si los dos confiesan el delito, tendrán que pasar cinco años en la cárcel por robo, pero si los dos no confiesan, sólo serán condenados a un año de cárcel por el delito, menos grave, de tenencia de artículos robados. Sin embargo, la policía ofrece a cada uno un trato: si uno testifica contra el otro, señalándole como cabecilla del robo,

el que testifique será puesto en libertad, mientras que el acusado pasará 10 años en la cárcel. Bajo estas condiciones, Alicia y Roberto se enfrentan al denominado **dilema del prisionero**: ¿debería testificar o rechazar el trato? Siendo agentes racionales, tanto Alicia como Roberto querrán maximizar sus propias utilidades esperadas. Supóngase que Alicia es cruelmente indiferente respecto al destino de su compañero y que, por lo tanto, su utilidad decrece en proporción con el número de años que pase en la cárcel, independientemente de lo que le suceda a Roberto. Roberto piensa exactamente de la misma forma. Para ayudar a alcanzar una decisión racional, ambos construyen la siguiente matriz de balances finales:

DILEMA DEL PRISIONERO

	Alicia: testificar	*Alicia: rechazar*
Roberto: testificar	$A = -5,\ R = -5$	$A = -10,\ \ R = 0$
Roberto: rechazar	$A = 0,\ \ R = -10$	$A = -1,\ R = -1$

Alicia analiza la matriz de balances finales como sigue: suponiendo que Roberto testifique, entonces pasaré cinco años en la cárcel si testifico y 10 si no lo hago por lo que, en este caso, testificar es mejor. Por otro lado, si Roberto rechaza el trato, entonces no iré a la cárcel si testifico y estaré un año si rechazo el trato por lo que, en este caso, también es mejor testificar. Por lo tanto, en cualquier caso, lo mejor para mí es testificar, por lo que debo hacerlo.

ESTRATEGIA DOMINANTE

DOMINACIÓN FUERTE

DOMINACIÓN DÉBIL

Alicia ha descubierto que testificar es la **estrategia dominante** para el juego. Se dice que una estrategia e para un jugador j **domina en sentido fuerte** a una estrategia e' si el resultado de e es mejor para j que el resultado de e', para cada elección de estrategias por parte de los otros jugadores. La estrategia e **domina en sentido débil** a e' si s es mejor que e' en al menos un perfil de estrategia y no es peor, en cualquier otro. Una estrategia dominante es una estrategia que domina a todas las otras. Es irracional tanto jugar con una estrategia dominada en sentido fuerte, como no jugar con una estrategia dominante, si ésta existe. Siendo racional, Alicia elige la estrategia dominante. Se necesita ampliar la terminología antes de continuar. Se dice que un resultado es un **óptimo de Pareto**[6] si no hay otro resultado que los jugadores preferirían. Un resultado es **dominado en sentido Pareto** por otro, si todos los jugadores prefiriesen este último.

ÓPTIMO DE PARETO

DOMINACIÓN EN SENTIDO PARETO

Si Alicia es «lista» a la vez que racional, debería continuar razonando de la siguiente forma: la estrategia dominante de Roberto también es la de testificar. Por lo tanto, él testificará y ambos serán condenados a cinco años. Cuando cada jugador tiene una estrategia dominante, la combinación de esas estrategias se denomina **equilibrio de estrategias dominantes**. En general, un perfil de estrategias constituye un **equilibrio** si ningún jugador puede beneficiarse por el hecho de cambiar de estrategia, conociendo que los demás jugadores actúan con la misma estrategia. Esencialmente, el equilibrio se alcanza en un **óptimo local** del espacio de políticas; es la cima de un pico a partir del cual la pendiente desciende a lo largo de cada dimensión (dimensiones que se corresponden con las posibles elecciones de estrategia de los jugadores).

EQUILIBRIO DE ESTRATEGIAS DOMINANTES

EQUILIBRIO

El *dilema* del prisionero es que el resultado para los dos jugadores en el punto de equilibrio es peor que el resultado que obtendrían si los dos rechazaran testificar. En otras

[6] La optimalidad de Pareto se denomina así en honor al economista Vilfredo Pareto (1848-1923).

palabras, el resultado para la solución de equilibrio está dominada en sentido Pareto por el resultado $(-1, -1)$ de (*rechazar*, *rechazar*).

¿Existe algún modo de que Alicia y Roberto lleguen al resultado $(-1, -1)$? Ciertamente, una opción *beneficiosa* para los dos es rechazar testificar, pero es una opción *inverosímil*. Cualquier jugador que contemple la posibilidad de *rechazar* se dará cuenta de que haría mejor si eligiese *testificar*. Este es el poder de atracción de un punto de equilibrio.

EQUILIBRIO DE NASH

El matemático John Nash (1928−) demostró que *cada juego tiene un equilibrio del tipo aquí definido*. Se conoce como **equilibrio de Nash** en su honor. Claramente, un equilibrio de una estrategia dominante es un equilibrio de Nash (Ejercicio 17.9), pero no todos los juegos tienen una estrategia dominante. El teorema de Nash viene a decir que hay estrategias de equilibrio, incluso cuando no hay una estrategia dominante.

Para el dilema del prisionero, sólo el perfil de estrategia (*testificar*, *testificar*) es un equilibrio de Nash. Es difícil entender cómo dos jugadores racionales pueden evitar este resultado ya que en cualquier solución propuesta de no equilibrio, al menos uno de los dos jugadores estará tentado a cambiar de estrategia. Los teóricos del juego están de acuerdo en que el equilibrio de Nash es una condición necesaria para que sea una solución (aunque no comparten que sea suficiente).

Es bastante fácil evitar la solución (*testificar*, *testificar*) si se cambia el juego (o los jugadores) de alguna forma. Por ejemplo, podría cambiarse a una versión de juego iterativa en la que los jugadores conociesen si volverán a encontrarse (pero con el detalle importante de que no sabrían cuántas veces más volverían a hacerlo). O si uno de los agentes tuviera convicciones morales que alentaran su espíritu de cooperación y justicia, podría cambiarse la matriz de balances finales de modo que reflejase la utilidad de cada agente a la hora de cooperar con el otro. Más adelante se verá que limitar la potencia de cálculo de los agentes, más que afectar a la capacidad de razonar de forma racional, puede afectar también al resultado.

Ahora se presenta un juego que no tiene una estrategia dominante. Acme, un fabricante de *hardware* para videojuegos, tiene que decidir si su próxima consola de juegos utilizará DVD o CD. Entretanto, el fabricante de *software* para videojuegos Best necesita decidir si produce su próximo juego para DVD o CD. Los beneficios de ambos serán positivos si están de acuerdo y negativos si están en desacuerdo, tal y como se muestra en la matriz de balances finales siguiente:

	Acme: dvd	*Acme: cd*
Best: dvd	$A = 9,\ B = 9$	$A = -4,\ B = -1$
Best: cd	$A = -3,\ B = -1$	$A = 5,\ B = 5$

No hay un equilibrio de estrategia dominante para este juego, pero hay dos equilibrios de Nash: (*dvd*, *dvd*) y (*cd*, *cd*). Se sabe que son equilibrios de Nash porque si cualquiera de los jugadores, de forma unilateral, cambiase a una estrategia diferente, este jugador terminaría peor. Ahora los agentes tienen un problema: *hay múltiples soluciones aceptables, pero si cada agente elige una solución diferente entonces el perfil de estrategia resultante no será una buena solución y ambos agentes padecerán esta situación*. ¿Cómo pueden acordar una solución? Una respuesta es que ambos deberían elegir la so-

lución óptima en sentido Pareto (*dvd*, *dvd*), es decir, pueden restringir la definición de «solución» al único equilibrio de Nash para el óptimo de Pareto, *siempre que exista uno*. Cada juego tiene al menos una solución óptima en sentido Pareto, pero puede tener varias o puede que no sean puntos de equilibrio. Por ejemplo, podría fijarse que el beneficio final de (*dvd*, *dvd*) fuera 5 en lugar de 9. En este caso, se tienen dos puntos iguales de equilibrio óptimos en sentido Pareto. Para elegir uno de ellos, los agentes pueden optar por adivinarlo o por *comunicarse*, bien mediante el establecimiento de una convención que ordene las soluciones, con anterioridad a que tenga lugar el juego, o bien mediante una negociación, en el transcurso del juego (que se traduciría en la inclusión de acciones de comunicación como parte de un juego de múltiples movimientos), que permita alcanzar un solución mutuamente beneficiosa. Por lo tanto, la comunicación en la teoría de juegos aparece exactamente por las mismas razones que apareció en la planificación multiagente vista en el Capítulo 12. Los juegos en los que los jugadores necesitan comunicarse de esta forma se denominan **juegos de coordinación**.

JUEGO DE COORDINACIÓN

Se ha visto que un juego puede tener más de un equilibrio de Nash, ¿cómo se sabe que cada juego debe tener al menos uno?, ¿el juego debe tener necesariamente uno? Puede ser que un juego no tenga equilibrios de Nash para las *estrategias puras*. Considérese, por ejemplo, un perfil de estrategia pura para el juego «pares o nones» (Sección 17.6). Si el número total de dedos es par entonces I querrá cambiar; si el total es impar entonces será P quien quiera cambiar. Por lo tanto, ningún perfil de estrategia pura puede ser un equilibrio y deben contemplarse las estrategias mixtas.

JUEGOS DE SUMA CERO

Pero ¿qué estrategia mixta? En 1928, von Neumann desarrolló un método para encontrar la estrategia mixta *óptima* para **juegos de suma cero** de dos jugadores. Un juego de suma cero es un juego en el que el balance en cada celda de la matriz de balances finales suman cero[7]. Claramente, «pares o nones» es un juego de este tipo. Para el caso de juegos de suma cero de dos jugadores, se sabe que los balances son iguales y opuestos, por lo que sólo se necesitará considerar los balances para un jugador que será el jugador a maximizar (igual que en el Capítulo 6). Para este juego, se elige al jugador impar I como jugador que maximiza, pudiéndose definir la matriz de balances finales por los valores $U_P(p, i)$ (el balance de P si muestra p dedos e I muestra i).

MAXIMIN

El método de von Neumann se denomina técnica **maximin** y funciona como sigue:

- Supóngase que cambiamos las reglas para forzar a P a revelar su estrategia en primer lugar, seguido después por I. Entonces se tiene un juego por turnos al cual puede aplicarse el algoritmo estándar **minimax** visto en el Capítulo 6. Supóngase que se tiene un resultado $U_{P,I}$. Claramente, este juego favorece a I por lo que la utilidad real U del juego (desde el punto de vista de P) es *al menos* $U_{P,I}$. Por ejemplo, si se contemplan estrategias puras, el árbol minimax del juego tiene un valor raíz de -3 (*véase* Figura 17.11(a)), por lo que se sabe que $U \geq -3$.

[7] Más general es el concepto de **juegos de suma constante** en los que cada balance suma una cantidad constante c. Un juego de suma constante de n jugadores puede transformarse en un juego de suma cero mediante la resta del valor c/n de cada balance. De este modo el ajedrez, con el balance tradicional de 1 para una victoria, 1/2 para el caso de quedar en tablas y 0 para una derrota, técnicamente, es un juego de suma constante con $c = 1$, pero puede transformarse fácilmente en un juego de suma cero mediante la sustracción de 1/2 a cada balance.

- Ahora supóngase que se cambian las reglas para forzar a I a revelar su estrategia primero y después P. Entonces el valor minimax de este juego es $U_{I,P}$ y dado que el juego favorece a P, se sabe que U es *al menos* $U_{I,P}$. Con estrategias puras, el valor es de $+2$ (*véase* Figura 17.11(b)), por lo que se sabe que $U \leq +2$.

Combinando estos dos argumentos, se tiene que la utilidad real U de la solución debe satisfacer

$$U_{P,I} \leq U \leq U_{I,P} \qquad \text{o en este caso} \qquad -3 \leq U \leq 2$$

Para determinar exactamente el valor de U, es necesario replantear el análisis desde el punto de vista de las estrategias mixtas. Primero, obsérvese que: *una vez que el primer jugador revela su estrategia, el segundo jugador no puede perder por jugar una estrategia pura*. La razón es sencilla: si el segundo jugador juega una estrategia mixta, [p: *uno*; $(1 - p)$: *dos*], su utilidad esperada es una combinación lineal $(p \cdot u_{uno} + (1 - p) \cdot u_{dos})$ de las utilidades de las estrategias puras u_{uno} y u_{dos}. Esta combinación lineal nunca puede ser mejor que la mejor de u_{uno} y u_{dos}, por lo que el segundo jugador podría jugar también con una estrategia pura.

Con esta observación en mente, puede pensarse que los árboles minimax tienen infinitas ramas en la raíz, que se corresponden con las infinitas estrategias mixtas que el primer jugador puede elegir. Cada una de ellas conduce a un nodo con dos ramas que se corresponden a las estrategias puras del segundo jugador. Estos infinitos árboles pueden representarse de forma finita si se utiliza una elección «parametrizada» en la raíz:

- Si P juega primero, la situación se muestra en la Figura 17.11(c). P juega [p: *uno*; $(1 - p)$: *dos*] en la raíz y entonces I elige un movimiento dado el valor de p. Si I elige *uno* el balance esperado (para P) es $2p - 3(1 - p) = 5p - 3$; si I elige *dos*, el balance esperado es $-3p + 4(1 - p) = 4 - 7p$. Estos balances pueden representarse como rectas en una gráfica donde p varía entre 0 y 1 sobre el eje de abscisas, tal y como se muestra en la Figura 17.11(e). I, el jugador que minimiza, siempre elegirá la menor de las dos rectas (mostradas en trazo grueso en la figura). Por lo tanto, lo mejor que P puede hacer en la raíz es elegir que p sea el probabilidad en el punto de intersección, es decir

  $$5p - 3 = 4 - 7p \qquad \Rightarrow \qquad p = 7/12$$

  La utilidad para P en este punto es $U_{P,I} = -1/12$.
- Si I mueve primero, la situación se muestra en la Figura 17.11(d). I juega [q: *uno*; $(1 - q)$: *dos*] en la raíz y entonces P elige un movimiento dado el valor de q. Los balances son $2q - 3(1 - q) = 5q - 3$ y $-3q + 4(1 - q) = 4 - 7q$[8]. Una vez más, la Figura 17.11(f) muestra que lo mejor que I puede hacer en la raíz es elegir la probabilidad en punto de intersección:

  $$5q - 3 = 4 - 7q \qquad \Rightarrow \qquad q = 7/12$$

  La utilidad para P en este punto es $U_{I,P} = -1/12$.

[8] Es una coincidencia que estas ecuaciones sean las mismas que para p; la coincidencia se debe a que $U_P(uno, dos) = U_P(dos, uno) = -3$. Esto explica también por qué la estrategia óptima es la misma para ambos jugadores.

Figura 17.11 (a) y (b): árboles minimax para el juego «pares o nones» si los jugadores juegan por turno estrategias puras. (c) y (d): árboles parametrizados del juego en los que el primer jugador juega una estrategia mixta. Los balances finales dependen de la probabilidad del parámetro (p o q) de la estrategia mixta. (e) y (f): para cada valor particular de probabilidad, el segundo deberá elegir la «mejor» de las dos acciones, de forma que el valor de la estrategia mixta del primer jugador viene dada por los tramos gruesos. El primer jugador elegirá como valor del parámetro la probabilidad de la estrategia mixta en el punto de intersección.

Así pues, se tiene que la utilidad real del juego está entre $-1/12$ y $-1/12$, es decir, que es exactamente $-1/12$. (La moraleja es que es mejor ser *I* que *P* si usted juega a este juego). Además, la utilidad real se alcanza mediante la estrategia mixta [$7/12$: *uno*; $5/12$: *dos*] a la que deberían jugar los dos jugadores. Esta estrategia se denomina **equilibrio maxi-**

EQUILIBRIO MAXIMIN

min del juego y es un equilibrio de Nash. Nótese que cada estrategia componente de la estrategia mixta en equilibrio tiene la misma utilidad esperada. En este caso, tanto *uno* como *dos* tienen la misma utilidad esperada $-1/12$, al igual que la propia estrategia mixta.

Este resultado para el juego «pares o nones» es un ejemplo del resultado general obtenido por von Neumann: *todo juego de suma cero con dos jugadores tiene un equilibrio maximin cuando se permiten estrategias mixtas*. Además, cada equilibrio de Nash en un juego de suma cero es un equilibrio maximin para dos jugadores. El algoritmo general para encontrar los equilibrios maximin en juegos de suma cero es un tanto más complicado de lo que pueden sugerir las Figuras 17.11(e) y (f). Cuando son n las posibles acciones, una estrategia mixta es un punto del espacio n-dimensional y las rectas se convierten en hiperplanos. También es posible que otras estrategias dominen alguna estrategia pura para el segundo jugador de modo que no sean óptimas frente a *cualquier* estrategia del primer jugador. Después de eliminar tales estrategias (que puede hacerse de forma repetida), la elección óptima en la raíz es un ejemplo de un problema de **programación lineal**: maximizar una función lineal sujeta a restricciones lineales. Tales problemas pueden resolverse por técnicas convencionales en un tiempo polinomial, dependiendo del número de acciones (y del número de bits utilizados por la función de recompensa, si se quiere ser estricto).

La pregunta crucial se mantiene, ¿qué debería hacer realmente un agente racional para jugar al juego «pares o nones» en su planteamiento inicial? El agente racional habrá derivado el hecho que [$7/12$: *uno*; $5/12$: *dos*] es la estrategia del equilibrio maximin, y supondrá que este conocimiento lo comparte el agente oponente. El agente podría utilizar un dado de 12 caras o un generador de números aleatorios para seleccionar aleatoriamente una acción de acuerdo con esta estrategia mixta, en cuyo caso el balance esperado sería $-1/12$ para P. O el agente sólo podría decidir jugar *uno* o *dos*. En cualquier caso, el balance esperado se mantiene en $-1/12$ para P. Curiosamente, la elección unilateral de una acción concreta no perjudica el balance esperado, pero permitir que el otro agente conozca con antelación la decisión unilateral adoptada por el primero, *sí* afecta al balance esperado puesto que el oponente puede ajustar consecuentemente su estrategia.

Encontrar las soluciones de juegos finitos de suma cero (es decir, equilibrios de Nash) es un poco más complicado. La aproximación general tiene dos pasos: (1) enumerar todos los posibles subconjuntos de acciones que podrían constituir estrategias mixtas. Por ejemplo, tratar primero todos los perfiles de estrategia donde cada jugador utiliza una única acción, después aquellas donde cada jugador utiliza una o dos acciones y así, sucesivamente. (2) para cada perfil de estrategia enumerado en el paso (1), comprobar si es un equilibrio. Esto se hace mediante la resolución de un conjunto de ecuaciones e inecuaciones que son semejantes a las utilizadas en el caso de suma cero. Para dos jugadores, estas ecuaciones son lineales y pueden resolverse con técnicas básicas de programación lineal, pero para tres o más jugadores no son lineales y puede ser muy complicado resolverlas.

JUEGOS REITERATIVOS

Hasta ahora se han estudiado tan sólo los juegos de un único movimiento. El tipo más sencillo de juegos con múltiples movimientos son los denominados **juegos reiterativos** en los que los jugadores se enfrentan a la misma elección reiteradamente, pero en todo momento, con conocimiento de los movimientos previos de todos los jugado-

PERFIL DE ESTRATEGIA DE UN JUEGO REITERATIVO

res. Un **perfil de estrategia** para un juego reiterativo especifica la elección de una acción para cada jugador en cada instante de tiempo para todas las historias posibles de movimientos anteriores. Al igual que los procesos de decisión de Markov, los balances son aditivos a lo largo del tiempo.

Considérese la versión reiterativa del dilema del prisionero. ¿Colaborarán Alicia y Roberto y rechazarán testificar, sabiendo que volverán a encontrarse de nuevo? La respuesta depende de los detalles. Por ejemplo, supóngase que Alicia y Roberto saben que ellos deberán jugar exactamente 100 rondas del dilema del prisionero. Entonces, sabrán que la última ronda no será un juego reiterativo (es decir, el resultado puede no tener efecto alguno sobre rondas futuras) y, por lo tanto, elegirán la estrategia dominante *testificar* en esa ronda. Una vez determinada la última ronda, la penúltima pasa también a no tener efecto sobre las siguientes rondas por lo que también elegirán el equilibrio de estrategia dominante, (*testificar*, *testificar*). Por inducción, ambos jugadores elegirán *testificar* en cada ronda, acumulando una pena de prisión de 500 años cada uno.

Pueden obtenerse diferentes soluciones si se cambian las reglas del juego. Por ejemplo, supóngase que, después de cada ronda, hay un 99 por ciento de probabilidades de que los dos jugadores vuelvan a encontrarse. Entonces, el número esperado de rondas es también de 100, pero ninguno de los jugadores conoce con total certeza cuál será la última ronda. Bajo estas condiciones, es posible un comportamiento más cooperativo. Por ejemplo, una estrategia de equilibrio para cada jugador es *rechazar*, a menos que el otro jugador haya elegido alguna vez *testificar*. Esta estrategia podría denominarse **castigo perpetuo**. Supóngase que ambos jugadores adoptan la misma estrategia y que ambos conocen este hecho. En tanto en cuanto ninguno de los jugadores haya elegido *testificar*, el balance total futuro esperado por cada jugador en cualquier instante es

$$\sum_{t=0}^{\infty} 0{,}99^t \cdot (-1) = -100$$

Un jugador que elija *testificar* obtendrá en el siguiente movimiento una puntuación de 0 en lugar de -1, si bien su balance futuro esperado pasa a ser

$$0 + \sum_{t=1}^{\infty} 0{,}99^t \cdot (-10) = -99$$

Por lo tanto, en cada paso, no hay ningún incentivo para desviarse de (*rechazar*, *rechazar*). El castigo perpetuo es la estrategia de «destrucción mutua asegurada» del dilema del prisionero: una vez que cualquiera de los jugador decide *testificar*, se garantiza que ambos jugadores lo lamentarán mucho. Pero esto sólo funciona como una medida disuasoria cuando el otro jugador cree que ha elegido esta estrategia (o cuando menos que puede haberla adoptado).

OJO POR OJO

Hay otras estrategias que son más indulgentes. La más famosa, denominada **ojo por ojo**, requiere comenzar con *rechazar* y después replicar la elección previa hecha por el otro jugador en el resto de movimientos. De este modo Alicia rechazaría el trato en tanto en cuanto Roberto también lo hiciera y testificaría después de que Roberto lo hubiera hecho y volvería a rechazarlo si Roberto lo hiciera. Aunque muy sencilla, se ha probado que esta estrategia es muy robusta y efectiva frente a una gran variedad de estrategias.

Pueden obtenerse también soluciones diferentes cambiando a los agentes en vez de cambiar las reglas del juego. Supóngase que los agentes son máquinas de estado finito con n estados y que están jugando un juego de m pasos, con $m > n$ pasos. Así pues, los agentes son incapaces de representar el número de pasos restantes y puede considerarse que este número es desconocido. Por lo tanto no pueden realizar el proceso de inducción anterior y son libres de alcanzar un equilibrio (*rechazar*, *rechazar*) más favorable. En este caso, la ignorancia *es* beneficiosa (o más bien, que tu oponente crea que eres ignorante es beneficioso). En estos juegos reiterativos, el éxito de uno depende de la *percepción* que otro oponente tenga sobre uno (si nos consideran un matón o un simplón) y no de características reales de uno.

En su completa extensión, los juegos reiterativos están fuera del alcance de este libro pero tienen lugar en muchos entornos. Por ejemplo, puede construirse un juego secuencial poniendo a dos agentes en el mundo de 4×3 celdas de la Figura 17.1. Si se especifica que no ocurre movimiento alguno cuando los dos agentes intentan moverse simultáneamente a la misma casilla (un problema habitual en muchos cruces de tráfico) entonces ciertas estrategias puras podrán bloquearse indefinidamente. Una solución es que cada agente, aleatoriamente, decida entre moverse o quedarse quieto; la situación de colapso se resolverá rápidamente y ambos agentes estarán contentos. Esto es exactamente lo que se hace para resolver las colisiones en las redes Ethernet.

En la actualidad, los métodos de solución conocidos para juegos reiterativos se asemejan a los algoritmos de juegos por turnos, vistos en el Capítulo 6, en los que el árbol de juego puede construirse de la raíz hacia abajo y resolverse de las hojas hacia arriba. La mayor diferencia es que, en lugar de considerar simplemente el máximo o mínimo de los valores de los hijos, el algoritmo debe resolver, en cada nivel, un juego con estrategias mixtas considerando que los nodos hijos se han resuelto y tienen valores conocidos con los que trabajar.

INFORMACIÓN PARCIAL

Los juegos reiterativos en entornos *parcialmente observables* se conocen como juegos de **información parcial**. Ejemplos de esta clase de juegos son los juegos de cartas como el póker o el *bridge*, en los que cada jugador sólo puede ver lar cartas de un subconjunto, y «juegos» más serios, como las abstracciones de la guerra nuclear, donde ningún bando conoce la localización de todas las armas de su oponente. Los juegos de información parcial se resuelven mediante la consideración de un árbol de estados de creencia, al igual que los procesos de decisión parcialmente observables (*véase* Sección 17.4). Una diferencia importante es que, mientras que el estado de creencia de uno es observable, el estado de creencia del oponente no lo es. Sólo recientemente se han desarrollado algoritmos prácticos para esta clase de juegos. Algunas versiones simplificadas del juego del póker se han resuelto, comprobándose que los «faroles» son realmente una elección racional, como parte de una estrategia mixta bien compensada. Una perspicacia importante que surge de tales estudios es que las estrategias mixtas son útiles, no sólo por hacer las acciones de uno impredecibles, sino también por minimizar la cantidad de información que el oponente puede aprender de la observación de nuestras acciones. Es interesante que, aunque los diseñadores de programas para jugar al *bridge* son conscientes de la importancia de recopilar y ocultar información, ninguno de ellos ha propuesto aún el uso de estrategias aleatorias.

Hasta ahora, ha habido algunas barreras para aplicar ampliamente la teoría de juegos en el diseño de agentes. En primer lugar, nótese que en una solución de equilibrio de Nash, un jugador está asumiendo que los oponentes jugarán definitivamente la estrategia de equilibrio. Esto significa que el jugador es incapaz de incorporar cualquier creencia que pueda tener acerca de cómo actuarán, probablemente, otros jugadores y, por lo tanto, que puede estar desperdiciando recursos defendiéndose frente amenazas que nunca se materializarán. La noción de **equilibrio en sentido Bayes-Nash** afronta parcialmente esta cuestión: es una solución de equilibrio con respecto a la distribución de probabilidad a priori de un jugador sobre las estrategias de los otros jugadores, es decir, expresa las creencias de un jugador sobre las estrategias probables de otros jugadores. En segundo lugar, no se dispone en la actualidad de mecanismos aceptables que permitan combinar las estrategias de la teoría de juegos con las estrategias de control para un proceso de decisión de Markov parcialmente observable. A causa de estos problemas, la teoría de juegos se ha utilizado principalmente para *analizar* entornos que están en equilibrio, más que para *controlar* agentes dentro de un entorno. Pronto se verá cómo puede ayudar a *diseñar* entornos.

EQUILIBRIO BAYES-NASH

17.7 Diseño de mecanismos

La sección anterior se centraba en la cuestión de que «conocido un juego, ¿en qué consiste una estrategia racional?». En esta sección se trata de determinar «¿cómo debería diseñarse un juego?, una vez conocido que son agentes racionales». Más específicamente, se desearía diseñar un juego cuyas soluciones, compuestas por las soluciones que cada agente obtendría con base en su propia estrategia, son resultado de maximizar alguna función de utilidad global. Este problema se denomina **diseño de mecanismos**, o en algunas ocasiones **teoría inversa de juegos**. El diseño de mecanismos es una materia básica de las ciencias económicas y políticas. Para grupos de agentes, se tiene la posibilidad de utilizar mecanismos basados en la teoría de juegos para construir sistemas inteligentes al margen de una colección de sistemas más limitados (incluso sistemas no cooperativos), de la misma forma que los seres humanos pueden alcanzar objetivos más allá de los límites individuales.

DISEÑO DE MECANISMOS

Ejemplos de diseños de mecanismos incluyen la subasta de billetes de avión baratos, el enrutado de paquetes TCP entre computadores, decidir cómo asignar médicos residentes a los hospitales y decidir cómo un robot que juegue al fútbol cooperará con sus compañeros de equipo. El diseño de mecanismos pasó a ser más que una materia académica, cuando en la década de los 90, varias naciones afrontaron el problema de subastar licencias de emisión en el espectro radioeléctrico, perdiendo cientos de millones de dólares en los ingresos potenciales como consecuencia de un diseño pobre de los mecanismos implicados. Formalmente, un **mecanismo** consta de (1) un lenguaje para describir el conjunto (posiblemente infinito) de estrategias permisibles que los agentes pueden adoptar y (2) la regla resultante G que determina los balances para los agentes, dado un perfil de estrategia de las estrategias permisibles.

MECANISMO

A primera vista, el problema del diseño de mecanismos puede parecer trivial. Supóngase que la función de utilidad global U se descompone en un conjunto de funciones de utilidad individuales de cada agente U_i, de tal modo que $U = \sum_i U_i$. Entonces, uno puede decir que si cada agente maximiza su propia utilidad, seguramente esto conducirá automáticamente a maximizar la utilidad global. (Por ejemplo, en la obra *El Capital* se dice que si todo el mundo trata de enriquecerse, la riqueza total de la sociedad se incrementará). Desafortunadamente, esto no es así. Las acciones de cada agente pueden afectar al bienestar de los otros agentes de tal forma que se decremente la utilidad global. Un ejemplo de esta situación es la **tragedia de los mancomunados**, una situación en la que todos los ganaderos, de forma individual, llevan a pastar su ganado a la dehesa comunal, con el resultado de la sobreexplotación de los pastos mancomunados y una utilidad negativa para todos los ganaderos. Cada ganadero, a título individual, actúa racionalmente, razonando que el uso de los pastos comunales es libre y que, aunque la sobreexplotación de éstos podría conducir a su destrucción, abstenerse de utilizarlos no le ayudaría nada (ya que los otros mancomunados lo utilizarían de todos modos). Una argumentación similar puede utilizarse para el vertido incontrolado de productos contaminantes a la atmósfera o a los océanos.

TRAGEDIA DE LOS MANCOMUNADOS

Una aproximación estándar en el diseño de mecanismos que aborda estos problemas es cobrar a cada agente por el uso de los bienes comunes. De forma más general, se necesita garantizar que todas las **influencias externas** (efectos sobre la utilidad global que no son reconocidas en las transacciones individuales de los agentes) se expliciten. El correcto establecimiento de los precios es una cuestión complicada. En el límite, esta aproximación viene a ser lo mismo que crear un mecanismo en el que se requiere efectivamente que cada agente maximice una utilidad global. Esta es una tarea extremadamente difícil para el agente, ya que ni puede valorar el estado actual del mundo ni observar los efectos de sus acciones sobre el resto de agentes. Por lo tanto, el diseño de mecanismos se concentra en encontrar mecanismos para los cuales los problemas de decisión a los que se enfrentan los agentes sean sencillos.

INFLUENCIAS EXTERNAS

Se considera primero el caso de las subastas. En su forma más habitual, una subasta es un mecanismo de venta de ciertas mercancías a una serie de compradores. Las estrategias son las pujas y los resultados determinan quién adquiere las mercancías y la cantidad de dinero pagada. Un ejemplo de dónde encajan las subastas dentro de la IA se tiene cuando un grupo de agentes deciden si cooperan o no en la realización de un plan conjunto. Hunsberger y Grosz (2000) mostraron que esta tarea puede realizarse de forma eficiente con una subasta en la que los agentes pujan por el papel jugado en la realización del plan.

Por ahora, se considerarán subastas en las que (1) hay una única mercancía, (2) cada pujador tiene un valor de utilidad v_i para la mercancía y (3) estos valores sólo son conocidos por el pujador. Los pujadores hacen pujas de valor b_i, y la mayor oferta obtiene la mercancía, pero el mecanismo determina cómo deben hacerse las pujas y el precio pagado por el ganador (no necesariamente b_i). El tipo de subasta mejor conocido es la **subasta inglesa** en la que los subastadores incrementan el precio de las mercancías y comprueban si los pujadores están aún interesados y así, hasta que sólo quede uno. Este mecanismo tiene la propiedad de que el pujador con el mayor valor v_i obtiene las mercancías por un precio de $b_m + d$, donde b_m es la mayor puja entre todos los jugadores y

SUBASTA INGLESA

d es el incremento que fija el subastador entre sucesivas ofertas[9]. La subasta inglesa también tiene la propiedad de que los pujadores tienen una estrategia dominante sencilla: mantener la puja, en tanto en cuanto, el coste actual esté por debajo de su valor personal. Conviene recordar que «dominante» quiere decir que la estrategia funciona bien frente a otra estrategia cualquiera, lo cual se traduce en que cada jugador puede adoptarla sin considerar las del resto. Por lo tanto, los jugadores no tienen que malgastar tiempo y energía contemplando las estrategias posibles de los otros jugadores. Un mecanismo en el que los jugadores tienen una estrategia dominante que implica revelar al resto sus verdaderas intenciones se denomina **de estrategia y prueba**.

ESTRATEGIA Y PRUEBA

Una propiedad negativa de la subasta inglesa es su elevado coste de comunicación ya que o bien la subasta tiene lugar en una sala de subastas o bien todos los pujadores disponen de líneas de comunicación seguras y de alta velocidad. Un mecanismo alternativo que requiere una comunicación menor es la **subasta con oferta cerrada**. En este caso, cada pujador hace una oferta única y se la comunica al subastador de forma que la mayor gana. Con este mecanismo, la estrategia de pujar por el valor verdadero no es dominante. Si su valor es v_i y usted cree que el máximo del resto de ofertas será b_m, entonces usted debería pujar por un valor que fuera el mínimo entre v_i y $b_m + \epsilon$. Dos de los inconvenientes de la pujas cerradas son que el jugador con el mayor v_i puede no obtener las mercancías y que los jugadores deben consumir recursos en observar las estrategias del resto de jugadores.

SUBASTA CON OFERTA CERRADA

Un pequeño cambio en las reglas de las subastas de oferta cerrada da lugar a la **subasta con oferta cerrada del segundo precio**, también conocida como **subasta Vickrey**[10]. En este tipo de subastas, el ganador paga el precio de la *segunda* puja más alta en vez de pagar su propia oferta. Esta sencilla modificación elimina las complejas deliberaciones necesarias para pujar en las subastas con ofertas cerradas convencionales (o del **primer precio**) ya que, ahora, la estrategia dominante es pujar por el valor actual que uno tiene. Para ver esto, hacemos notar que cada jugador puede pensar que la subasta es como un juego de dos jugadores si se ignora al resto de jugadores, con la excepción de él mismo y el pujador con la mayor oferta. La utilidad del jugador i, en función de su oferta b_i, su valor v_i y la mayor oferta entre el resto de ofertas del resto de jugadores, b_m, es

SUBASTA CON OFERTA CERRADA DEL SEGUNDO PRECIO

SUBASTA VICKREY

$$u_i(b_i, b_m) = \begin{cases} (v_i - b_m) & \text{si } b_i > b_m \\ 0 & \text{en otro caso} \end{cases}$$

Para ver que $b_i = v_i$ es una estrategia dominante, nótese que cuando $(v_i - b_m)$ es positivo, cualquier oferta que gane la subasta es óptima, y pujar por v_i, en concreto, gana la subasta. Por otro lado, cuando $(v_i - b_m)$ es negativo, cualquier oferta que pierda la subasta es óptima y, en concreto, pujar por v_i pierde la subasta. Por lo tanto, pujar por v_i es el óptimo para todos los posibles valores de b_m y, de hecho, v_i es la única oferta que tiene esta propiedad. Por su simplicidad y escasos recursos computacionales necesarios,

[9] En realidad existe una pequeña probabilidad de que el jugador con el mayor valor v_i no adquiera las mercancías en el caso de que $b_m < v_i < b_m + d$. La probabilidad de que esto ocurra puede hacerse tan pequeña como se quiera, disminuyendo el valor de d.

[10] Denominada así, en honor a Willian Vickrey (1914−1996), ganador del premio Nobel de Economía en 1996.

tanto para el vendedor como para los pujadores, la subasta Vickrey se utiliza ampliamente en la construcción de sistemas de IA distribuidos.

Considérese ahora el problema del enrutado en Internet. Los jugadores son ahora las aristas del grafo de conexiones de red. Cada jugador conoce el coste c_i de enviar un mensaje a través del enlace; el coste de no tener que enviar un mensaje es 0. El objetivo es encontrar la ruta más barata que debe seguir un mensaje desde un origen a un destino, donde el coste de la ruta completa es la suma de los costes individuales de cada enlace. En el Capítulo 4 se vieron varios ejemplos para calcular la ruta más corta dado el coste de las arista de manera que lo que tiene que hacerse es solicitar a cada agente que informe sobre su coste real c_i. Desafortunadamente, si sólo se pregunta esto a cada agente, se obtendrán informes con costes elevados para que se aliente a enviar el mensaje por otro enlace. Se necesita, pues, un mecanismo a prueba de estrategias. Este mecanismo consiste en pagar a cada jugador una compensación p_i igual a la longitud de la ruta más corta que no contenga el i-ésimo enlace menos la longitud de la ruta más corta suponiendo que el coste a través del i-ésimo enlace es 0:

$$p_i = \textsc{Longitud}(\text{ruta con } c_i = \infty) - \textsc{Longitud}(\text{ruta con } c_i = 0)$$

Puede demostrarse que, de acuerdo con este mecanismo, la estrategia dominante de cada jugador es la de informar fielmente sobre c_i y que, procediendo de esta forma, se obtendrá la ruta más barata. A pesar de esta propiedad, el mecanismo esbozado no se utiliza en la práctica debido al elevado coste de comunicación y de computación. El diseñador del mecanismo debe comunicarse con los n jugadores y debe resolver el problema de optimización. Esto podría merecer la pena si los costes pudieran amortizarse considerando todos los mensajes, pero en una red real, los costes c_i fluctuarían constantemente debido a la congestión en el tráfico y porque algunas máquinas pueden caer, mientras que otras pueden comenzar a funcionar. Todavía no se ha concebido una solución, completamente satisfactoria, para este problema.

17.8 Resumen

Este capítulo muestra cómo utilizar el conocimiento acerca del mundo para adoptar decisiones, incluso cuando los resultados de una acción son inciertos y las recompensas por actuar no pueden obtenerse hasta que no hayan transcurrido un número suficiente de acciones. Los puntos principales son los siguientes:

- Los problemas de decisión secuenciales en entornos con incertidumbre, también llamados **procesos de decisión de Markov** (PDM) se definen mediante un **modelo de transición**, que especifica los resultados probabilistas de cada acción y una **función de recompensa**, que determina la recompensa en cada estado.
- La utilidad de una secuencia de estados viene dada por la suma de todas las posibles recompensas sobre la secuencia y posiblemente depreciadas a lo largo del tiempo. La solución de un PDM es una **política** que asocia una decisión a cada estado que el agente puede alcanzar. Una política óptima maximiza la utilidad de las secuencias de estados encontradas cuando ésta se ejecuta.

- La utilidad de un estado es la utilidad esperada de las secuencias de estados encontradas cuando se ejecuta una política óptima partiendo de ese estado. El algoritmo de **iteración de valores** para la resolución de procesos de decisión de Markov se basa en la resolución iterativa de las ecuaciones que relacionan las utilidades de cada estado con las de sus vecinos.
- La **iteración de políticas** alterna entre el cálculo de utilidades de los estados de acuerdo con la política actual y la mejora de la política actual en relación con las utilidades actuales.
- Los procesos de decisión de Markov parcialmente observables (PDMPO) son mucho más difíciles de resolver que los procesos de decisión de Markov. Pueden resolverse mediante su conversión a un PDM en el espacio continuo de estados de creencia. El comportamiento óptimo en procesos de decisión parcialmente observables incluye la recopilación de información para reducir la incertidumbre y, por tanto, tomar mejores decisiones en el futuro.
- Puede construirse un agente basado en la teoría de la decisión para entornos de tipo PDMPO. El agente utiliza una **red de decisión dinámica** para representar los modelos de transición y observación, modificar su estado de creencia y proyectarlo hacia el futuro sobre las posibles secuencias de acciones.
- La **teoría de juegos** describe el comportamiento racional de los agentes en situaciones en las que interactúan simultáneamente varios agentes. Las soluciones de los juegos son **equilibrios de Nash** (perfiles de estrategia en los que ningún agente tiene incentivo alguno en desviarse de la estrategia especificada).
- El **diseño de mecanismos** puede usarse para establecer las reglas mediante las cuales los agentes interactuarán, con el fin de maximizar alguna utilidad global a través de la operación, a título individual, de agentes racionales. En algunas ocasiones existen mecanismos que alcanzan este objetivo sin necesidad de que cada agente considere las elecciones hechas por otros agentes.

Se retomará el mundo de los procesos de decisión de Markov, convencionales y observables parcialmente, en el Capítulo 21, cuando se estudien los métodos de **aprendizaje por refuerzo** que permiten a un agente mejorar su comportamiento a partir de la experiencia en entornos secuenciales y con incertidumbre.

Notas bibliográficas e históricas

Richard Bellman (1957) inició la aproximación moderna a los problemas de decisión secuenciales y propuso la aproximación propia de la programación dinámica como solución general, proponiendo en particular el algoritmo de iteración de valores. La tesis doctoral de Ron Howard (1960) introdujo la iteración de políticas y la idea de recompensa media para la resolución de problemas con un horizonte infinito. Bellman y Dreyfus (1962) aportaron varios resultados adicionales. La modificación de la iteración de políticas se debe a van Nunen (1976) y a Puterman y Shin (1978). Williams y Baird (1993) analizaron la iteración de políticas asíncrona y demostraron también la cota de

pérdida de la política dada por la Ecuación (17.9). El análisis del descuento en función de las preferencias estacionarias se debe a Koopmans (1972). Los textos de Bertsekas (1987), Puterman (1994) y Bertsekas y Tsitsiklis (1987) proporcionan una rigurosa introducción a los problemas de decisión secuenciales. Papadimitriou y Tsitsiklis (1987) describen resultados sobre la complejidad computacional de los procesos de decisión de Markov.

Los trabajos fundamentales de Sutton (1988) y Watkins (1989) sobre métodos de aprendizaje por refuerzo para la resolución de procesos de decisión de Markov jugaron un papel relevante en la introducción de los procesos de decisión de Markov en la comunidad de la IA, al igual que lo hizo posteriormente el estudio de Barto *et al*. (1995). (Un trabajo anterior de Werbos (1977) contenía muchas ideas similares pero no alcanzó la misma extensión.) La conexión entre procesos de decisión de Markov y los problemas de planificación en IA se debe en primer lugar a Sven Koenig (1991), que demostró cómo los operadores probabilistas de STRIPS proporcionaban una representación concisa para los modelos de transición (*véase* también Wellman (1990b)). Los intentos de utilización de un horizonte de búsqueda limitado y de estados abstractos para superar el problema de la explosión combinatoria en grandes espacios de estados se deben a Dean *et al*. (1993) y Tash y Russell (1994). Las heurísticas basadas en el valor de información pueden emplearse para seleccionar áreas del espacio de estados en los que una expansión local del horizonte ocasionará una mejora significativa en la calidad de la decisión. Los agentes que utilizan esta aproximación pueden adaptar su esfuerzo para tratar con la presión temporal y generar algunos comportamientos interesantes tales como el uso de «rutas marcadas» para encontrar sus caminos alrededor del espacio de estados rápidamente sin tener que recalcular las decisiones óptimas en cada punto. Un trabajo reciente de Boutilier y otros (Boutilier *et al*., 2000, 2001) se ha centrado en la programación dinámica utilizando representaciones *simbólicas*, tanto de los modelos de transición como de las funciones de valor, con base en la formulación proposicional y de primer orden.

La observación de que un MPD parcialmente observable puede transformarse en un MPD convencional mediante el uso de estados de creencia se debe a Astrom (1965). El primer algoritmo completo para la solución exacta de procesos de decisión de Markov parcialmente observables fue propuesta por Edward Sondik (1971) en su tesis doctoral. (Un artículo posterior de Smallwood y Sondik (1973) presenta algunos errores, pero es más comprensible.) Lovejoy (1991) estudió el estado del arte de los procesos de decisión de Markov parcialmente observables. La primera contribución significativa dentro de la IA fue el algoritmo Witness (Cassandra *et al*., 1994; Kaelbling *et al*., 1998), una versión mejorada de la iteración de valores para PDMPO. Pronto surgieron otros algoritmos entre los que se encuentran la aproximación de Hansen (1998), que construye incrementalmente una política en forma de autómata de estados finito. Con esta representación de la política, el estado de creencia se corresponde directamente con un estado del autómata. Mediante una búsqueda anticipada y, en combinación con un muestreo de las observaciones posibles y los resultados de las acciones, pueden construirse de forma aproximada soluciones óptimas para un PDMPO (Kearns *et al*., 2000; Ng y Jordan, 2000). El Capítulo 21 contiene trabajos adicionales sobre algoritmos de resolución de procesos de decisión parcialmente observables.

Dean y Kanazawa (1989a) propusieron las ideas básicas de una arquitectura de agentes basada en el uso de redes de decisión dinámicas. El libro de Dean y Wellman (1991) *Planning and Control* profundiza mucho más, estableciendo conexiones entre modelos como las redes Bayesianas y de decisión dinámicas con la literatura sobre filtrado de la teoría de control clásica. Tatman y Shachter (1990) mostraron cómo aplicar algoritmos de programación dinámica a las redes de decisión dinámicas. Russell (1998) explica varias vías en que estos agentes pueden ampliarse e identifica varias líneas de investigación abiertas.

Las raíces de la teoría de juegos pueden establecerse con las propuestas hechas en el siglo XVII por Christiaan Huygens y Gottfried Leibniz para estudiar interacciones humanas de carácter cooperativo y competitivo de forma científica y matemática. A lo largo del siglo XIX, varios economistas crearon ejemplos matemáticos sencillos para analizar ejemplos concretos de situaciones competitivas. Los primeros resultados formales de la teoría de juegos se deben a Zermelo (1913) (que había sugerido un año antes una forma de búsqueda minimax para juegos, aunque incorrecta). Emile Borel (1921) introdujo la notación de una estrategia mixta. John von Neumann (1928) probó que todo juego de suma cero para dos jugadores tiene un equilibrio maximin dentro de las estrategias mixtas y un valor bien definido. La colaboración de von Neumann con el economista Oskar Morgenstern condujo a la publicación en 1944 del libro *Theory of Games and Economic Behavior* que sienta las bases de la teoría de juegos. La publicación del libro se retrasó por la escasez de papel en tiempos de la Segunda Guerra Mundial hasta que un miembro de la familia Rockefeller subvencionó personalmente su publicación.

En 1950, a la edad de 21 años, John Nash publicó sus ideas relativas a equilibrios en juegos. Su definición de una solución en equilibrio, aunque originaria de Cournot (1838), pasó a ser conocida como equilibrio Nash. Después de un largo período de tiempo debido a la esquizofrenia que sufrió a partir de 1959, Nash fue galardonado con el premio Nobel en Economía (junto con Reinhart Selten y John Harsanyi) en 1994.

Harsanyi (1967) describió el equilibrio de Bayes-Nash que posteriormente fue analizado por Kadane y Larkey (1982). Binmore (1982) trata acerca de algunas cuestiones referidas al uso de la teoría de juegos en el control de agentes.

El dilema del prisionero fue inventado como un ejercicio de clase por Albert W. Tucker en 1950 y fue analizado ampliamente por Axelrod (1985). Luce y Raiffa (1957) introdujeron los juegos reiterativos, mientras que Kuhn (1953) introdujo los juegos de información parcial. El primer algoritmo práctico para juegos con información parcial fue desarrollado por Koller *et al*. (1996) en el ámbito de la IA; el artículo de Koller y Pfeffer (1997) proporciona una introducción amena a esta área y describe un sistema de trabajo para representar y resolver juegos secuenciales. La teoría de juegos y los procesos de decisión de Markov se combinan en la teoría de juegos de Markov (Littman, 1994). Realmente, Shapley (1953) describió antes que Bellman el algoritmo de iteración de valores, pero sus resultados no fueron ampliamente apreciados, quizá porque se presentaron en el contexto de los juegos de Markov. Entre los libros acerca de la teoría de juegos incluyen los escritos por Myerson (1991), Fudenberg y Tirole (1991) y Osborne y Rubinstein (1994).

La tragedia de los mancomunados, problema que motiva el diseño de mecanismos, fue presentado por Hardin (1968). Hurwicz (1973) creó los fundamentos matemáticos para el diseño de mecanismos. Milgrom (1997) escribió sobre la subasta del espectro electromagnético que diseñó. Las subastas pueden emplearse en planificación (Hunsberger y Grosz, 2000) y planificación de procesos (Rassenti *et al*., 1982). Varian (1995) da una pequeña introducción con referencias a la literatura informática, y Rosenschein y Zlotkin (1994) son autores de un extenso libro que incluye aplicaciones en la IA distribuida. Otros trabajos sobre IA distribuida incluyen trabajos sobre Inteligencia Colectiva (Tumer y Wolpert, 2000) y control basado en mercados (Clearwater, 1996). Artículos sobre cuestiones computacionales de las subastas aparecen a menudo en la publicación ACM *Conferences on Electronic Commerce*.

EJERCICIOS

17.1 Para el mundo de 4×3 celdas mostrado en la Figura 17.1, calcule qué celdas puede alcanzar la secuencia de acciones [*Arriba*, *Arriba*, *Derecha*, *Derecha*, *Derecha*] si se parte de la posición (1, 1) y con qué probabilidades. Explique cómo se relaciona este cálculo con la tarea de proyectar un modelo de Markov oculto.

17.2 Suponga que definimos la utilidad de una secuencia de estados como la *máxima* recompensa obtenida en cualquiera de los estados de la secuencia. Demuestre que esta función de utilidad no tiene solución para las preferencias estacionarias entre secuencias de estados. ¿Es posible definir una función de utilidad sobre los estados tal que, la toma de decisión de acuerdo con el principio de MUE de un comportamiento óptimo?

17.3 ¿Puede cualquier problema de búsqueda finita trasladarse exactamente a un problema de decisión de Markov de tal forma que una solución en este último problema sea solución óptima del primero? Si es así, explique *precisamente* cómo transformar el problema y cómo transformar la solución. Si no es así, explique *precisamente* por qué no (es decir, de un contraejemplo).

17.4 Considere un PDM sin descuento que tiene tres estados (1, 2, 3) con recompensas -1, -2 y 0, respectivamente. El estado 3 es un estado terminal. En los estados 1 y 2 se tienen dos posibles acciones: a y b. El modelo de transición es el siguiente:

- En el estado 1, la acción a mueve al agente al estado 2 con probabilidad 0,8 y hace permanecer al agente en la misma situación con probabilidad 0,2.
- En el estado 2, la acción a mueve al agente al estado 1 con probabilidad 0,8 y hace permanecer al agente en la misma situación con probabilidad 0,2.
- En cualquiera de los estados 1 y 2, la acción b mueve al agente al estado 3 con probabilidad 0,1 y mantiene al agente en el mismo estado con una probabilidad de 0,9.

Conteste a las siguientes preguntas.

a) *Cualitativamente*, ¿qué puede decirse acerca de la política óptima en los estados 1 y 2?

b) Aplique la iteración de políticas, mostrando por completo cada paso, para establecer la política óptima y los valores de los estados 1 y 2. Suponga que la política inicial tiene la acción b en ambos estados.

c) ¿Qué sucede en la iteración de políticas si la política inicial tiene la acción a en ambos estados? ¿Depende la política óptima de un factor de descuento?

17.5 En algunas ocasiones, los procesos de decisión de Markov se formulan empleando una función de recompensa $R(s, a)$ que depende de la acción tomada o una función de recompensa $R(s, a, s')$ que también depende del estado resultante.

a) Escriba las ecuaciones de Bellman para estos casos.

b) Muestre cómo un PDM con función de recompensa $R(s, a, s')$ puede transformarse en un PDM diferente con función de recompensa $R(s, a)$ de tal forma que las políticas óptimas del nuevo PDM se corresponden exactamente con las políticas del PDM original

c) Ahora haga lo mismo para convertir un PDM con $R(s, a)$ en un PDM con $R(s)$.

17.6 Considere el mundo de 4×3 mostrado en la Figura 17.1

a) Implemente un simulador de este entorno de tal forma que pueda cambiarse fácilmente la geografía específica del entorno. Parte del código para hacer esto ya está disponible en el repositorio de código *online*.

b) Diseñe un agente que utilice la iteración de políticas y mida su rendimiento en el simulador del entorno a partir de varios estados de partida. Realice varios experimentos a partir de cada estado de partida y compare la recompensa total media obtenida por ejecución con la utilidad del estado, también determinada por el algoritmo.

c) Experimente incrementando el tamaño del entorno. ¿Cómo varía el tiempo de ejecución de la iteración de políticas con el tamaño del entorno?

17.7 Para el entorno mostrado en la Figura 17.1 encuentre todos los valores umbral de $R(s)$ tal que la política óptima cambie cuando se crucen estos umbrales. Necesitará una forma de calcular la política óptima y su valor para una recompensa fija $R(s)$. [*Pista*: pruebe que el valor de cualquier política prefijada varía linealmente con $R(s)$].

17.8 En este ejercicio consideraremos el proceso de decisión de Markov de dos jugadores que se corresponden con los juegos por turnos de suma cero como los vistos en el Capítulo 6. Sean A y B los jugadores y sea $R(s)$ la recompensa para el jugador A en el estado s. (La recompensa de B es siempre igual y opuesta.)

a) Sea $U_A(s)$ la utilidad del estado s cuando A tiene el turno para mover en el estado s, y sea $U_B(s)$ la utilidad de s cuando le corresponde a B el turno de mover en s. Todas las recompensas y utilidades se calculan desde el punto de vista de A (al igual que en el árbol minimax de juegos). Escriba las ecuaciones de Bellman que definen $U_A(s)$ y $U_B(s)$.

b) Explique cómo realizar la iteración de valores para dos jugadores con estas ecuaciones y defina un criterio de parada apropiado.

c) Considere el juego descrito en la Figura 6.14. Dibuje el espacio de estados (mejor que el árbol del juego) mostrando los movimientos de A con líneas sólidas

y los movimientos de B con líneas discontinuas. Marque cada estado con $R(s)$. Encontrará útil organizar los estados (s_A, s_B) en una malla bidimensional usando s_A y s_B como «coordenadas.»

d) Aplique ahora la iteración de valores de dos jugadores para resolver este juego y derive/obtenga la política óptima.

17.9 Muestre que un equilibrio de estrategia dominante es un equilibrio de Nash, pero no a la inversa.

17.10 En el juego de niños roca-papel-tijeras cada jugador muestra al mismo tiempo el objeto elegido entre roca, papel o tijeras. Los papeles envuelven a las rocas, las rocas embotan a las tijeras y las tijeras cortan papel. En la versión extendida roca-papel-tijeras-fuego-agua, el fuego gana a la roca, el papel y las tijeras; la roca, el papel y las tijeras ganan al agua, y el agua gana al fuego. Escriba la matriz de balances finales y encuentre una solución de estrategia mixta para este juego.

17.11 Resuelva el juego «pares o nones» con tres dedos.

17.12 Antes de 1999, los equipos de la Liga Nacional de Hockey recibían dos puntos por una victoria, uno por un empate y cero por una derrota. ¿Es un juego de suma constante? En 1999 se modificaron las reglas de forma que un equipo que pierda en la prórroga recibe un punto. El equipo ganador continúa obteniendo dos puntos. ¿Cómo cambia la respuesta a la pregunta anterior con estos cambios? Si fuera del todo legal, ¿cuándo sería razonable que los dos equipos amañaran el partido para terminar empatados el partido en el tiempo reglamentario, para después disputar realmente la victoria en la prórroga? Suponga que la utilidad de cada equipo es el número de puntos que recibe y que existe una probabilidad a priori p de que el primer equipo gane en la prórroga (conocida por los dos equipos). ¿Para qué valores de p podrían llegar ambos equipos a este acuerdo?

17.13 La siguiente matriz de balances finales, tomada de Blinder (1983) por parte de Bernstein (1996), muestra un juego entre políticos y la Reserva Federal de Estados Unidos.

	Fed: reducir	Fed: no hacer nada	Fed: aumentar
Pol: reducir	$F = 7,\ P = 1$	$F = 9,\ P = 4$	$F = 6,\ P = 6$
Pol: no hacer nada	$F = 8,\ P = 2$	$F = 5,\ P = 5$	$F = 4,\ P = 9$
Pol: aumentar	$F = 3,\ P = 3$	$F = 2,\ P = 7$	$F = 1,\ P = 8$

Los políticos pueden aumentar o disminuir la presión fiscal mientras que la Reserva Federal puede controlar la política monetaria (y por supuesto cada una de las partes puede no hacer nada). Cada parte tiene también preferencias acerca de quién debería adoptar las medidas (ninguna de las partes quiere aparecer ante la opinión pública como «el malo de la película»). Los balances mostrados son simplemente una ordenación: 9 para la primera elección hasta 1 para la última elección. Encuentre el equilibrio de Nash para el juego en las estrategias puras. ¿Es esta una solución óptima en sentido Pareto? El lector debería desear analizar las políticas de las administraciones recientes en este sentido.

18 Aprendizaje de observaciones

En este capítulo se describen agentes que pueden mejorar su comportamiento estudiando sus propias experiencias.

La idea del aprendizaje consiste en utilizar las percepciones no sólo para actuar, sino también para mejorar la habilidad del agente para actuar en el futuro. El aprendizaje entra en juego cuando el agente observa sus interacciones con el mundo y sus procesos de toma de decisiones. Puede ir desde la trivial memorización de las experiencias, como mostraba el agente del mundo de *wumpus* en el Capítulo 10, hasta la creación de teorías científicas, como mostró Albert Einstein. Este capítulo describe el **aprendizaje inductivo** a partir de observaciones. En particular, se describe cómo aprender teorías sencillas en lógica proposicional. Además, se proporciona un análisis teórico que explica por qué el aprendizaje inductivo funciona bien.

18.1 Formas de aprendizaje

En el Capítulo 2, vimos que un agente de aprendizaje puede ser diseñado con un **elemento de acción,** que decide qué acciones llevar a cabo y con un **elemento de aprendizaje,** que modifica el elemento de acción para poder tomar mejores decisiones. (*Véase* Figura 2.15). Los investigadores en el campo del aprendizaje han creado una gran variedad de elementos de aprendizaje. Para comprenderlos, se mostrará cómo su diseño se ve afectado por el contexto en el que operan. El diseño de un elemento de aprendizaje se ve afectado mayoritariamente por tres aspectos:

- Qué *componentes* del elemento de acción tienen que aprenderse.
- Qué *realimentación* está disponible para aprender dichos componentes.
- Qué tipo de *representación* se usa para los componentes.

A continuación se van a analizar cada uno de estos aspectos. Como se ha mostrado, existen muchas formas de construir el elemento de acción de un agente. En el Capítulo 2 se describen varios diseños de agentes (Figuras 2.9, 2.11, 2.13 y 2.14). Los componentes de estos agentes incluyen lo siguiente:

1. Una proyección directa de las condiciones del estado actual a las acciones.
2. Un método para inferir las propiedades relevantes del mundo a partir de una secuencia de percepciones.
3. Información sobre cómo evoluciona el mundo y sobre los resultados de las posibles acciones que el agente puede llevar a cabo.
4. Información de utilidad, que indica lo deseables que son los estados.
5. Información acción-valor, que indica lo deseables que son las acciones.
6. Metas que describen las clases de estados que maximizan la utilidad del agente.

Cada uno de estos componentes puede aprenderse con una realimentación apropiada. Consideremos, por ejemplo, un agente entrenándose para convertirse en un taxista. Cada vez que el instructor grite «¡Frene!», el agente puede aprender una regla condición-acción que le indique cuándo frenar (componente 1). Informando al agente de que ciertas imágenes de cámaras contienen autobuses, puede aprender a reconocerlos (2). Realizando acciones y observando sus resultados (por ejemplo frenar bruscamente en pavimento mojado) puede aprender las consecuencias de sus acciones (3). Además, si no recibe propina de pasajeros tras una conducción brusca, puede aprender una componente útil de su función de utilidad (4).

El *tipo de realimentación* disponible para el aprendizaje normalmente es el factor más importante a la hora de determinar la naturaleza del problema de aprendizaje que tiene que afrontar el agente. Se distinguen tres distintos tipos de aprendizaje: **supervisado**, **no supervisado** y por **refuerzo**.

APRENDIZAJE SUPERVISADO

El problema de **aprendizaje supervisado** consiste en aprender una función a partir de ejemplos de sus entradas y sus salidas. Los casos (1), (2) y (3) son ejemplos de problemas de aprendizaje supervisado. En el caso (1), el agente aprende la regla condición-acción para frenar, esto es, una función que a partir del estado genera una salida booleana (frenar o no frenar). En el caso (2), el agente aprende una función que a partir de una imagen genera una salida booleana (si la imagen contiene o no un autobús). En el caso (3), aprende una función que a partir del estado y las acciones para frenar, genera la distancia de parada expresada en pies. Nótese que tanto en los casos (1) y (2), un profesor suministra el valor correcto de la salida de cada ejemplo; en el tercero, el valor de la salida proviene de lo que el agente percibe. En entornos totalmente observables, el agente siempre puede observar los efectos de sus acciones, y por lo tanto, puede utilizar métodos de aprendizaje supervisado para aprender a predecirlos. En entornos que son parcialmente observables, el problema es más difícil, ya que los efectos más inmediatos pueden ser invisibles.

APRENDIZAJE NO SUPERVISADO

El problema de **aprendizaje no supervisado** consiste en aprender a partir de patrones de entradas para los que no se especifican los valores de sus salidas. Por ejemplo, un agente taxista debería desarrollar gradualmente los conceptos de «días de tráfico bueno» y de «días de tráfico malo», sin que le hayan sido dados ejemplos etiquetados de ello. Un agente de aprendizaje supervisado puro no puede aprender qué hacer, porque no tiene

información de lo que es una acción correcta o un estado deseable. Principalmente se estudiará aprendizaje no supervisado en el contexto de los sistemas de razonamiento probabilístico (Capítulo 20).

El problema del **aprendizaje por refuerzo**, que se describe en el Capítulo 21, es el más general de las tres categorías. En vez de que un profesor indique al agente qué hacer, el agente de aprendizaje por refuerzo debe aprender a partir del **refuerzo**[1]. Por ejemplo, la falta de propina al final del viaje (o una gran factura por golpear la parte trasera del coche de delante) da al agente algunas indicaciones de que su comportamiento no es el deseable. El aprendizaje por refuerzo típicamente incluye el subproblema de aprender cómo se comporta el entorno.

APRENDIZAJE POR REFUERZO

REFUERZO

La forma de *representar la información aprendida*, también representa un papel importante para determinar el algoritmo de aprendizaje. Cualquiera de los componentes de un agente puede ser representado usando cualquiera de los esquemas de representación mostrados en este libro. Se han mostrado varios ejemplos: polinomios lineales con peso para representar funciones de utilidad en programas de teoría de juegos; sentencias en lógica proposicional y de primer orden para todas las componentes de un agente lógico; y descripciones probabilísticas, como Redes Bayesianas, para las componentes de inferencia de un agente basado en la teoría de la decisión. Se han ideado algoritmos eficaces de aprendizaje para todos ellos. En este capítulo se mostrarán métodos para lógica proposicional, en el Capítulo 19 se describen métodos para lógica de primer orden, y en el Capítulo 20 se tratan métodos para Redes Bayesianas y para redes de neuronas (que incluyen polinomios lineales como un caso especial).

El factor más importante en el diseño de sistemas de aprendizaje es la *disponibilidad de conocimiento a priori*. La mayoría de los investigadores de aprendizaje en IA, ingeniería informática y psicología han estudiado el caso en el que el agente comienza sin información sobre lo que está intentando aprender. Sólo tiene acceso a los ejemplos de sus experiencias. Aunque es un caso importante, no es por término medio el caso más general. La mayoría del aprendizaje humano tiene lugar en un contexto con bastante conocimiento de base. Algunos psicólogos y lingüistas reivindican que incluso los niños recién nacidos poseen conocimiento del mundo. Verdad o no, no hay duda de que el conocimiento previo puede ayudar en gran medida en el aprendizaje. Un físico examinando una pila de fotografías de una cámara de burbujas (*bubble-chamber*), podría ser capaz de inferir una teoría que afirmara la existencia de una nueva partícula con una cierta masa y una cierta carga; sin embargo un crítico de arte que examina la misma pila podría aprender únicamente que el «artista» debe pertenecer a alguna clase de impresionismo abstracto. El Capítulo 19 muestra varios casos en los que se facilita el aprendizaje gracias a la existencia de conocimiento; también se muestra cómo el conocimiento puede ser *recopilado* para acelerar el proceso de toma de decisiones. El Capítulo 20 muestra cómo el conocimiento previo ayuda en el aprendizaje de teorías probabilísticas.

[1] El término **recompensa,** que también fue utilizado en el Capítulo 17, es un sinónimo de **refuerzo**.

18.2 Aprendizaje inductivo

EJEMPLO

INFERENCIA INDUCTIVA PURA

HIPÓTESIS

GENERALIZACIÓN

PROBLEMA DE INDUCCIÓN

ESPACIO DE HIPÓTESIS

CONSISTENCIA

NAVAJA DE OCKHAM

Un algoritmo para aprendizaje supervisado determinístico recibe como entrada el valor correcto para determinados valores de una función desconocida y debe averiguar cuál es la función o aproximarla. Siendo más formales, se dice que un **ejemplo** es un par (x, $f(x)$), donde x es la entrada, y $f(x)$ es la salida de la función f aplicada a x. La tarea de la **inferencia inductiva pura** (o **inducción**) es la siguiente:

Dada una colección de ejemplos de f, devolver una función h que aproxime a f.

La función h se denomina **hipótesis**. La razón por la cual el aprendizaje es difícil, desde un punto de vista conceptual, es que no es fácil determinar si una función h es una buena aproximación de f. Una buena hipótesis estará bien **generalizada** si puede predecir ejemplos que no se conocen. Éste es el **problema de inducción** fundamental, que ha sido estudiado durante siglos; el Apartado 18.5 aporta una solución parcial.

La Figura 18.1 muestra un ejemplo típico: ajustar una función de una única variable a una serie de puntos dados. Los ejemplos son pares (x, $f(x)$), donde tanto x como $f(x)$ son números reales. Se elige el **espacio de hipótesis H** (el conjunto de hipótesis que se van a considerar) como el conjunto de polinomios de grado inferior o igual a k, por ejemplo $3x^2 + 2$, $x^{17} - 4x^3$. La Figura 18.1(a) muestra algunos datos para los que se puede ajustar de forma exacta una línea recta (un polinomio de grado 1). La línea se denomina hipótesis **consistente** ya que verifica todos los datos. La Figura 18.1(b) muestra un polinomio de mayor grado que también es consistente con los mismos datos. Esto ilustra la primera cuestión del aprendizaje inductivo: *¿Cómo elegir entre múltiples hipótesis consistentes?* Una respuesta la proporciona la **navaja de Ockham**[2]: es preferible la hipótesis consistente con los datos que sea más sencilla. Intuitivamente, tiene sentido, ya que las hipótesis que no son tan sencillas como los datos van a fallar a la hora de extraer cualquier *patrón* de los mismos. Definir qué es sencillo no es fácil, pero parece razonable decir que un polinomio de grado 1 es más sencillo que uno de grado 12.

Figura 18.1 (a) Ejemplos de pares (x, $f(x)$) y una hipótesis lineal consistente. (b) Un polinomio de grado 7 como hipótesis consistente para el mismo conjunto de datos. (c) Un conjunto de datos diferentes que admite un polinomio de grado 6 que ajusta de forma exacta o un encaje lineal aproximado. (d) Un encaje sinusoidal sencillo para el mismo conjunto de datos.

[2] Nombre del filósofo inglés del siglo XIV, William de Ockham. A veces se escribe como «Occam», quizá por la traducción francesa, «Guillaume d'Occam».

La Figura 18.1 (c) muestra un segundo conjunto de datos. No existe una línea recta consistente para este conjunto de datos; de hecho, requiere un polinomio de grado 6 (con siete parámetros) para un ajuste exacto. Los datos están formados por siete puntos, así que el polinomio tiene tantos parámetros como puntos forman los datos; por ello, no parece encontrarse ningún patrón en los datos y no esperamos una buena generalización. Sería mejor ajustar con una línea recta, que no sea exactamente consistente, pero que permita hacer predicciones razonables. Esto equivale a aceptar la posibilidad de que la función verdadera sea no determinística (o algo similar, que las entradas reales no han sido observadas en su totalidad). *Para funciones no deterministas, existe un inevitable compromiso entre la complejidad de la hipótesis y el grado de adecuación de los datos*. El Capítulo 20 explica cómo realizar este compromiso usando la teoría de la probabilidad.

Se debe tener en mente que la posibilidad o la imposibilidad de encontrar una hipótesis consistente y sencilla depende en gran medida del espacio de hipótesis elegido. La Figura 18.1 (d) muestra que los datos de (c) pueden ajustarse en su totalidad mediante una sencilla función de la forma $ax + b + c\ \mathrm{sen}(x)$. Este ejemplo muestra la importancia de la elección del espacio de hipótesis. Un espacio de hipótesis formado por polinomios de grado finito no puede representar funciones sinusoidales con precisión, así que no se puede aprender con este espacio de hipótesis a partir de datos sinusoidales. Se dice que un problema de aprendizaje es **realizable** si el espacio de hipótesis contiene a la función verdadera; en otro caso, se dice que es **irrealizable**. Desgraciadamente, no siempre se puede determinar si un problema de aprendizaje es realizable, ya que la función verdadera no se conoce. Una forma de evitar este problema es usar *conocimiento a priori* para elegir un espacio de hipótesis en el que sepamos que se encuentra la función verdadera. Este tema se detalla en el Capítulo 19.

REALIZABLE

IRREALIZABLE

Otro enfoque es utilizar un espacio de hipótesis tan grande como sea posible. Por ejemplo, ¿por qué no permitir que **H** sea el conjunto de todas las máquinas de Turing? Después de todo, toda función computable puede ser representada por una máquina de Turing. El problema de este enfoque es que no tiene en cuenta la complejidad computacional del aprendizaje. *Existe un compromiso entre la expresividad del espacio de hipótesis y la complejidad de encontrar hipótesis sencillas y consistentes dentro de este espacio*. Por ejemplo, ajustar líneas rectas a los datos es muy sencillo; ajustar polinomios de grado alto es más complicado; y ajustar máquinas de Turing es muy complicado, ya que dada una máquina de Turing, determinar si es consistente con los datos no es un problema, en general, decidible. Una segunda razón para preferir espacios de hipótesis simples es que la hipótesis resultante será sencilla de utilizar, es decir, será más rápido computar $h(x)$ cuando h sea una función lineal que cuando sea un programa de cualquier máquina de Turing.

Por estas razones, la mayoría de los trabajos en aprendizaje se han enfocado hacia representaciones relativamente simples. En este capítulo, nos concentramos en la lógica proposicional y lenguajes relacionados. El Capítulo 19 muestra la teoría de aprendizaje en lógica de primer orden. Se mostrará que el compromiso entre expresividad y complejidad no es tan sencillo como parece a simple vista; es muy típico el caso, como se vio en el Capítulo 8, de que un lenguaje expresivo haga posible una teoría *sencilla* para ajustarse a los datos, mientras que si se restringe la expresividad del lenguaje, cualquier otra teoría consistente será más compleja. Por ejemplo, las reglas para el ajedrez

pueden ser expresadas con lógica de primer orden con una o dos páginas, sin embargo se requieren miles de páginas cuando se expresan en lógica proposicional. En estos casos, debería ser posible aprender más rápido usando el lenguaje más expresivo.

18.3 Aprender árboles de decisión

La inducción con árboles de decisión es uno de los métodos más sencillos y con más éxito para construir algoritmos de aprendizaje. Sirve como una buena introducción al área de aprendizaje inductivo, y es muy sencillo de implementar. Primero, se describen los árboles de decisión como herramienta de desarrollo, y luego se muestra cómo aprenderlos. Mientras tanto, se introducen ideas que aparecen en todas las áreas de aprendizaje inductivo.

Árboles de decisión como herramienta de desarrollo

ÁRBOLES DE DECISIÓN

ATRIBUTOS

CLASIFICACIÓN

REGRESIÓN

POSITIVO

NEGATIVO

Un **árbol de decisión** toma como entrada un objeto o una situación descrita a través de un conjunto de **atributos** y devuelve una «decisión»: el valor previsto de la salida dada la entrada. Los atributos de entrada pueden ser discretos o continuos. A partir de ahora, asumiremos entradas discretas. El valor de la salida puede ser a su vez discreto o continuo; aprender una función de valores discretos se denomina **clasificación**; aprender una función continua se denomina **regresión**. Nos concentraremos en clasificaciones booleanas, en las cuales cada ejemplo se clasifica como verdadero (**positivo**) o falso (**negativo**).

Un árbol de decisión desarrolla una secuencia de test para poder alcanzar una decisión. Cada nodo interno del árbol corresponde con un test sobre el valor de una de las propiedades, y las ramas que salen del nodo están etiquetadas con los posibles valores de dicha propiedad. Cada nodo hoja del árbol representa el valor que ha de ser devuelto si dicho nodo hoja es alcanzado. La representación en forma de árboles de decisión es muy natural para los humanos; en realidad muchos manuales que explican cómo hacer determinadas tareas (por ejemplo, reparar un coche) están escritos en su totalidad como un único árbol de decisión abarcando cientos de páginas.

PREDICADO META

Un ejemplo sencillo se muestra en el problema de decidir si esperar por una mesa en un restaurante. Aquí, el propósito es aprender una definición para el **predicado meta** *Esperar*. Esto es un problema de aprendizaje, primero se tiene que establecer qué atributos están disponibles para describir los ejemplos del dominio. En el Capítulo 19 se verá cómo automatizar esta tarea; a partir de ahora, supondremos que vamos a decidir sobre la siguiente lista de atributos:

1. *Alternativa*: si existe un restaurante adecuado alternativo cerca.
2. *Bar*: si el restaurante tiene un área de bar confortable en la que se pueda esperar.
3. *Vier/Sab*: verdad si es viernes o sábado.
4. *Hambriento*: si estamos hambrientos.

5. *Clientes*: cuánta gente hay en el restaurante (puede tomar los valores: *Ninguno*, *Algunos* y *Lleno*).
6. *Precio*: el rango de precios del restaurante ($, $$, $$$).
7. *Lloviendo*: si está lloviendo fuera.
8. *Reserva*: si se ha hecho una reserva.
9. *Tipo*: la clase de restaurante (francés, italiano, tailandés, o hamburguesería).
10. *TiempoEsperaEstimado*: el tiempo de espera estimado por el dueño (0-10 minutos, 10-30 minutos, 30-60 minutos, >60).

En la Figura 18.2 se muestra el árbol de decisión que se ha utilizado. Téngase en cuenta que el árbol no usa los atributos *Precio* y *Tipo*, de hecho se les considera irrelevantes. Los ejemplos son procesados por el árbol empezando en el nodo raíz y continuando por la rama apropiada hasta que se llega a un nodo hoja. Por ejemplo dado *Clientes* = *Lleno* y *TiempoEsperaEstimado* = 0-10 se clasificará como positivo (es decir, sí, se esperará por una mesa).

Expresividad de los árboles de decisión

Desde el punto de vista de la lógica, cualquier hipótesis sobre el predicado meta *Esperar,* de un árbol de decisión particular, puede ser vista como una aserción de la forma:

$$\forall s \quad Esperar(s) \Leftrightarrow (P_1(s) \vee \cdots \vee P_n(s))$$

donde cada condición $P_i(s)$, es una conjunción de test que corresponde a un camino desde la raíz del árbol hasta una hoja con un resultado positivo. Aunque ésta parezca una sentencia de primer orden, de alguna forma es proposicional, porque contiene exacta-

Figura 18.2 Un árbol de decisión para decidir si esperar por una mesa.

mente una variable y todos los predicados son unarios. En realidad, el árbol de decisión describe una relación entre *Esperar* y algunas combinaciones lógicas de valores de atributos. No se puede utilizar un árbol de decisión para representar test que se refieran a dos o más objetos diferentes, por ejemplo,

$$\exists r_2 \quad Cerca(r_2, r) \wedge Precio(r, p) \wedge Precio(r_2, p_2) \wedge MasBarato(p_2, p)$$

(¿hay algún restaurante más barato cerca?). Obviamente, se podría añadir otro atributo booleano con el nombre *RestauranteMasBaratoCerca*, pero sería intratable añadir todos los atributos. En el Capítulo 19 investigaremos más sobre el problema de aprendizaje en la lógica de predicados propiamente dicha.

Los árboles de decisión pueden expresar exactamente lo mismo que los lenguajes de tipo proposicional; esto es, cualquier función Booleana puede ser escrita como un árbol de decisión. Esto se puede hacer de forma trivial, haciendo corresponder cada fila de la tabla de verdad con la función correspondiente a cada camino del árbol. Esto llevaría a un árbol de decisión con un crecimiento exponencial, ya que las filas de la tabla de verdad crecen de forma exponencial a medida que se introducen nuevos atributos. Obviamente, los árboles de decisión pueden representar muchas funciones con árboles mucho más pequeños.

FUNCIÓN DE PARIDAD

FUNCIÓN MAYORÍA

Para muchos tipos de funciones, sin embargo, esto es realmente un problema. Por ejemplo, si la función es la **función de paridad**, que devuelve 1 si y sólo si un número par de entradas es igual a 1, se necesitará un árbol de decisión de crecimiento exponencial. También trae dificultades el uso de un árbol de decisión para representar la **función mayoría**, que devuelve 1 si más de la mitad de las entradas son 1.

En otras palabras, los árboles de decisión son buenos para algunos tipos de funciones y malos para otros. ¿Existe *algún* tipo de representación que sea eficiente para *todos* los tipos de funciones? Desafortunadamente, la respuesta es no. Mostraremos esto de una forma general. Considere el conjunto de todas las funciones Booleanas sobre n atributos. ¿Cuántas funciones diferentes hay en este conjunto? Es exactamente el número de tablas de verdad diferentes que podemos escribir, porque una función se define por su tabla de verdad. La tabla de verdad tiene 2^n filas, ya que cada caso de entrada viene descrito por n atributos. Podemos considerar la columna «respuesta» de la tabla como un número de 2^n bits que define la función. No importa qué representación se utilice para la función, algunas funciones (de hecho, casi todas) requerirán muchos bits para representarlas.

Si se toman 2^n bits para definir la función, hay 2^{2^n} funciones diferentes sobre n atributos. Es un número espantoso. Por ejemplo, con seis atributos Booleanos, hay $2^{2^6} = 18$, 466, 744, 073, 709, 551, 616 funciones diferentes para elegir. Necesitaremos algún algoritmo ingenioso para encontrar hipótesis consistentes en un espacio tan grande.

Inducir árboles de decisión a partir de ejemplos

Un ejemplo para un árbol de decisión booleano consiste en un vector de atributos de entrada, X, y en un único valor de salida booleano y. Un conjunto de ejemplos (X_1, y_1) ..., (X_{12}, y_{12}) se muestra en la Figura 18.3.

Ejemplo	Atributos										Meta
	Alt	*Bar*	*Vier*	*Ham*	*Clientes*	*Precio*	*Llov*	*Res*	*Tipo*	*Est*	Esperar
X_1	*Sí*	*No*	*No*	*Sí*	*Algunos*	*$$$*	*No*	*Sí*	*Francés*	*0-10*	*Sí*
X_2	*Sí*	*No*	*No*	*Sí*	*Lleno*	*$*	*No*	*No*	*Tailandés*	*30-60*	*No*
X_3	*No*	*Sí*	*No*	*No*	*Algunos*	*$*	*No*	*No*	*Hamburg.*	*0-10*	*Sí*
X_4	*Sí*	*No*	*Sí*	*Sí*	*Lleno*	*$*	*Sí*	*No*	*Tailandés*	*10-30*	*Sí*
X_5	*Sí*	*No*	*Sí*	*No*	*Lleno*	*$$$*	*No*	*Sí*	*Francés*	*>60*	*No*
X_6	*No*	*Sí*	*No*	*Sí*	*Algunos*	*$$*	*Sí*	*Sí*	*Italiano*	*0-10*	*Sí*
X_7	*No*	*Sí*	*No*	*No*	*Ninguno*	*$*	*Sí*	*No*	*Hamburg.*	*0-10*	*No*
X_8	*No*	*No*	*No*	*Sí*	*Algunos*	*$$*	*Sí*	*Sí*	*Tailandés*	*0-10*	*Sí*
X_9	*No*	*Sí*	*Sí*	*No*	*Lleno*	*$*	*Sí*	*No*	*Hamburg.*	*>60*	*No*
X_{10}	*Sí*	*Sí*	*Sí*	*Sí*	*Lleno*	*$$$*	*No*	*Sí*	*Italiano*	*10-30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*Ninguno*	*$*	*No*	*No*	*Tailandés*	*0-10*	*No*
X_{12}	*Sí*	*Sí*	*Sí*	*Sí*	*Lleno*	*$*	*No*	*No*	*Hamburg.*	*30-60*	*Sí*

Figura 18.3 Ejemplos para el dominio del restaurante.

CONJUNTO DE ENTRENAMIENTO

Los ejemplos positivos son aquellos en los que la meta *Esperar* es verdadera ($X_1, X_3, ...$); los ejemplos negativos son aquellos en los que es falsa ($X_2, X_5, ...$). El conjunto de ejemplos completo se denomina **conjunto de entrenamiento**.

El problema de encontrar un árbol de decisión que sea consistente con el conjunto de entrenamiento puede parecer difícil, pero, de hecho, existe una solución trivial. Podríamos construir simplemente un árbol de decisión que tenga un camino hasta una hoja para cada ejemplo, donde el camino comprueba cada atributo y sigue el valor que tiene en el ejemplo hasta llegar a una hoja que contiene la clasificación del mismo. Cuando se proporciona el mismo ejemplo otra vez[3], el árbol de decisión propondrá la clasificación correcta. ¡Desafortunadamente, no tendrá mucho que decir sobre cualquier otro caso!

El problema con este árbol trivial es que memoriza exactamente las observaciones. No extrae ningún patrón a partir de los ejemplos, y por lo tanto no se puede esperar que sea capaz de extrapolar a ejemplos que no le han sido proporcionados. Aplicando la navaja de Ockham, deberíamos encontrar en vez de este árbol, el árbol de decisión *más pequeño* que sea consistente con los ejemplos. Desafortunadamente, para cualquier definición razonable de «el más pequeño», encontrar el árbol más pequeño es un problema intratable. Sin embargo, con algunas heurísticas simples, se puede hacer un buen trabajo y encontrar un árbol más bien pequeño. La idea básica del algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN es realizar primero el test sobre el atributo más importante. Se considera como «atributo más importante» aquel que discrimina más claramente los ejemplos. De esta forma, esperamos obtener la clasificación correcta con un número de test pequeño, es decir, que todos los caminos en el árbol sean cortos y así el árbol completo será pequeño.

[3] El mismo ejemplo *o un ejemplo con la misma descripción* (esta distinción es muy importante, volveremos a ella en el Capítulo 19).

La Figura 18.4 muestra cómo comienza el algoritmo. Hay 12 ejemplos de entrenamiento, que se clasifican en ejemplos positivos y negativos. Después, se decide qué atributo utilizar como primer test en el árbol. En la Figura 18.4(a) se observa que *Tipo* es un atributo pobre, porque nos deja con cuatro resultados posibles, cada uno de los cuales tiene el mismo número de ejemplos positivos y negativos. Por otra parte, en la Figura 18.4(b) se observa que *Clientes* es un atributo bastante importante, porque si el valor es *Ninguno* o *Algunos* obtenemos conjuntos de ejemplos para los que se puede responder definitivamente (*No* y *Sí*, respectivamente). Si el valor es *Completo*, obtenemos un conjunto de ejemplos mezclados. En general, después de que el primer atributo de test separe los ejemplos, cada resultado constituye en si mismo un nuevo problema de aprendizaje del árbol de decisión con menos ejemplos y un atributo menos. Hay cuatro casos a considerar para estos problemas recursivos:

1. Si quedan ejemplos positivos y negativos, hay que elegir el mejor atributo para separarlos. La Figura 18.4(b) muestra cómo se ha utilizado el atributo *Hambriento* para separar los ejemplos que quedan.
2. Si todos los ejemplos restantes son positivos (o todos negativos), se ha finalizado: se puede responder *Sí* o *No*. La Figura 18.4(b) muestra ejemplos de este caso para los valores *Ninguno* y *Algunos*.
3. Si no quedan ejemplos, significa que ningún ejemplo ha sido observado y se devuelve un valor por defecto calculado a partir de la clasificación de la mayoría en el nodo padre.
4. Si no quedan atributos, pero sí ejemplos positivos y negativos, tenemos un problema, ya que los ejemplos que quedan tienen exactamente la misma descripción, pero clasificaciones diferentes. Esto sucede cuando alguno de los datos es incorrecto; diremos que hay **ruido** en los datos. También ocurre cuando los atributos no proporcionan suficiente información para describir completamente la

RUIDO

Figura 18.4 Dividiendo los ejemplos mediante test sobre los atributos. (a) La división de *Tipo* no permite distinguir entre ejemplos negativos y positivos. (b) La división de *Clientes* hace un buen trabajo de separación entre ejemplos positivos y negativos. Después de *Clientes*, *Hambriento* es un atributo apropiado para el segundo test.

situación, o cuando el dominio no es determinista. Una forma sencilla de resolver el problema es utilizar el voto de la mayoría.

En la Figura 18.5 se muestra el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN. Los detalles del método ELEGIR-ATRIBUTO se proporcionan en la siguiente subsección.

En la Figura 18.6 se muestra el árbol final que se produce utilizando el conjunto de datos de los 12 ejemplos. El árbol es claramente distinto del árbol original de la Figura 18.2, a pesar de que en realidad, los datos fueron generados por un agente utilizando el árbol original. Se podría concluir que el algoritmo de aprendizaje no está realizando un buen trabajo en el aprendizaje de la función correcta. Sin embargo, ésta sería una con-

```
función APRENDIZAJE-ÁRBOL-DECISIÓN(ejemplos, atribs, por-defecto) devolver árbol de decisión
  entradas: ejemplos, conjunto de ejemplos
            atribs: conjunto de atributos
            por-defecto: valor por defecto del predicado meta

  si ejemplos está vacio entonces devolver por-defecto
  si no si todos los elementos de ejemplos tienen la misma clasificación entonces
     devolver la clasificación
  si no si atribs está vacio entonces devolver VALOR-MAYORÍA(ejemplos)
  si no
      mejor ← ELEGIR-ATRIBUTO(atribs, ejemplos)
      arbol ← un nuevo árbol de decisión con nodo raíz mejor
      m ← VALOR-MAYORÍA(ejemplos_i)
      para cada valor v_i de mejor hacer
          ejemplos_i ← {elementos de ejemplos con mejor = v_i}
          subarbol ← APRENDIZAJE-ÁRBOL-DECISIÓN(ejemplos_i,atribs-mejor,m)
          añadir una rama a arbol con la etiqueta v_i y el subárbol subarbol
      devolver arbol
```

Figura 18.5 El algoritmo de aprendizaje del árbol de decisión.

Figura 18.6 Árbol de decisión inducido a partir del conjunto de entrenamiento de 12 ejemplos.

clusión errónea. El algoritmo de aprendizaje recibe los *ejemplos*, no la función correcta, y de hecho, su hipótesis (*véase* Figura 18.6) no sólo es consistente con todos los ejemplos, sino que además es considerablemente más sencilla que el árbol original. El algoritmo de aprendizaje no tiene necesidad de incluir los test para *Lloviendo* y *Reserva*, porque puede clasificar todos los ejemplos sin ellos. Además, ha detectado un patrón insospechado: el primer autor esperará por comida tailandesa los fines de semana.

Por supuesto, si hubiéramos reunido más ejemplos, se habría inducido un árbol más similar al original. El árbol de la Figura 18.6 está destinado a cometer un error; por ejemplo, nunca se le ha presentado el caso de que el restaurante esté lleno y el tiempo de espera sea de 0-10 minutos. Para un caso donde *Hambriento* sea falso, el árbol dice que no hay que esperar, pero yo (SR), desde luego esperaría. Esto lleva a una cuestión obvia: si el algoritmo induce, a partir de los ejemplos, un árbol consistente pero incorrecto, habría que preguntarse en qué medida es incorrecto este árbol. Mostraremos cómo analizar esta pregunta de forma experimental, después de explicar los detalles del paso de selección de atributo.

Elección de los atributos de test

En aprendizaje de árboles de decisión, el esquema que se utiliza para seleccionar atributos está diseñado para minimizar la profundidad del árbol final. La idea es elegir el atributo que proporcione una clasificación lo más exacta posible de los ejemplos. Un atributo perfecto divide los ejemplos en conjuntos que contienen sólo ejemplos positivos o negativos. El atributo *Clientes* no es perfecto, pero es bastante adecuado. Un atributo inadecuado, como *Tipo*, deja el conjunto de ejemplos, aproximadamente, con la misma proporción de ejemplos positivos y negativos que el conjunto original.

Todo lo que necesitamos, por lo tanto, es una medida formal de «bastante adecuado» e «inadecuado», y podremos implementar la función ELEGIR-ATRIBUTO de la Figura 18.5. La medida debe tener su valor máximo cuando el atributo es perfecto, y su valor mínimo cuando el atributo es inadecuado. Una medida adecuada es la cantidad esperada de **información** proporcionada por el atributo, donde utilizamos el término en su sentido matemático, definido en primer lugar en Shannon y Weaver (1949). Para entender el concepto de información, se puede pensar en la respuesta a una pregunta, por ejemplo, si una moneda caerá de cara. La cantidad de información que contiene la respuesta depende del conocimiento que se tenga *a priori*, cuanto menos se sepa, más información proporcionará la respuesta. La teoría de la información mide el contenido de información en **bits**. Un bit de información es suficiente para responder una pregunta de tipo *sí/no* sobre la que no se sabe nada, como el resultado del lanzamiento de una moneda equilibrada. En general, si las respuestas posibles v_i tienen probabilidades $P(v_i)$, el contenido de información I de la respuesta actual viene dado por:

INFORMACIÓN

$$I\big(P(v_1), ..., P(v_n)\big) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i)$$

Para el caso del lanzamiento de una moneda equilibrada tendríamos:

$$I\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2}\log_2 \frac{1}{2} - \frac{1}{2}\log_2 \frac{1}{2} = 1 \text{ bit}$$

Si la moneda se manipula de forma que se obtenga cara el 99 por ciento de las veces que se lanza, tendríamos $I(1/100, 99/100) = 0{,}08$ bits, como la probabilidad de obtener cara tiende a 1, la información de la respuesta actual tiende a 0.

En el aprendizaje de árboles de decisión, nos preguntaremos cuál es la clasificación correcta de un ejemplo dado. Un árbol de decisión correcto podrá responder a esta pregunta. Las proporciones de ejemplos positivos y negativos del conjunto de entrenamiento proporcionan una estimación de las probabilidades de las posibles respuestas antes de que ningún atributo haya sido elegido para test. Suponiendo que el conjunto de entrenamiento contiene p ejemplos positivos y n ejemplos negativos, una estimación de la información contenida en la respuesta correcta sería:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

El conjunto de entrenamiento del ejemplo del restaurante de la Figura 18.3, tiene $p = n = 6$, por lo tanto, necesitamos un bit de información.

Un test sobre un atributo simple A no nos proporcionará, por lo general, mucha información, pero sí alguna. Podemos medir exactamente cuánta información proporciona, determinando cuánta información necesitaremos después de realizar el test. Cualquier atributo A divide el conjunto de entrenamiento E en subconjuntos $E_1,...E_v$ de acuerdo con sus valores para A, donde A puede tener v valores distintos. Cada subconjunto E_i tiene p_i ejemplos positivos y n_i ejemplos negativos, por lo tanto, si vamos por esta rama, necesitaremos $I(p_i/(p_i + n_i), n_i/(p_i + n_i))$ bits adicionales para responder la pregunta. Un ejemplo elegido aleatoriamente del conjunto de entrenamiento tiene el valor i-ésimo para el atributo con probabilidad $(p_i + n_i)/(p + n)$, por lo tanto, después de hacer el test sobre el atributo A, en general necesitaremos

$$Resto(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

GANANCIA DE INFORMACIÓN

bits de información para clasificar el ejemplo. La **ganancia de información** del atributo de test es la diferencia entre la necesidad de información original y la nueva necesidad de información:

$$\text{Ganancia}(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Resto}(A)$$

La heurística que se utiliza en la función ELEGIR-ATRIBUTO consiste en elegir el atributo con mayor ganancia. Volviendo a los atributos considerados en la Figura 18.4, tenemos:

$$Ganancia(Clientes) = 1 - \left[\frac{2}{12}I(0,1) + \frac{4}{12}I(1,0) + \frac{6}{12}I\left(\frac{2}{6}, \frac{4}{6}\right)\right] \approx 0,541 \quad bits$$

$$Ganancia(Tipo) = 1 - \left[\frac{2}{12}I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12}I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12}I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12}I\left(\frac{2}{4}, \frac{2}{4}\right)\right] = 0$$

Con lo que se confirma nuestra intuición de que el atributo *Clientes* es mejor para separar los ejemplos. De hecho, *Clientes* tiene mayor ganancia que cualquiera de los demás atributos que podrían ser elegidos como raíz por el algoritmo de aprendizaje de árboles de decisión.

Valoración de la calidad del algoritmo de aprendizaje

Un algoritmo de aprendizaje es bueno si produce hipótesis que hacen un buen trabajo al predecir clasificaciones de ejemplos que no han sido observados. En el Apartado 18.5 mostraremos cómo se puede estimar la calidad de una predicción por adelantado. De momento, veremos una metodología para calcular la calidad de una predicción después de que ocurra el hecho que se ha predicho.

Obviamente, una predicción es buena si resulta cierta, por lo tanto, se puede calcular la calidad de una hipótesis contrastando sus predicciones con la clasificación correcta, una vez se conoce ésta. Esto se hace con un conjunto de ejemplos denominado **conjunto de test**. Si entrenamos utilizando todos los ejemplos disponibles, después necesitaremos algunos ejemplos más para hacer el test, así que es más conveniente adoptar la siguiente metodología:

CONJUNTO DE TEST

1. Recolectar un conjunto de ejemplos grande.
2. Dividir el conjunto de ejemplos en dos conjuntos disjuntos: el conjunto de entrenamiento y el conjunto de test.
3. Aplicar el algoritmo de aprendizaje al conjunto de entrenamiento, generando la hipótesis *h*.
4. Medir el porcentaje de ejemplos del conjunto de test que *h* clasifica correctamente.
5. Repetir los pasos de 1 a 4 para conjuntos de entrenamiento de diferentes tamaños y conjuntos de entrenamiento seleccionados aleatoriamente para cada tamaño.

El resultado de este procedimiento es un conjunto de datos que puede ser procesado para obtener la calidad media de predicción como una función del tamaño del conjunto de entrenamiento. Si dibujamos esta función en una gráfica, obtendremos lo que se denomina **curva de aprendizaje** del algoritmo para cada dominio concreto. En la Figura 18.7 se muestra la curva de aprendizaje para el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN con el ejemplo del restaurante. Se puede observar que a medida que el conjunto de entrenamiento crece, la calidad de la predicción se incrementa. (Por esta razón, este tipo de curvas se llaman también **gráficas felices** (*happy graphs*)). Este es un buen síntoma de que, efectivamente, hay algún patrón en los datos y el algoritmo de aprendizaje está encontrándolo.

CURVA DE APRENDIZAJE

Obviamente, al algoritmo de aprendizaje no se le debe permitir «ver» los datos de test hasta que se utilizan para contrastar una hipótesis aprendida. Desafortunadamente, es muy fácil caer en la trampa de «ojear» (***peeking***) los datos de test. Esto típicamente sucede de la siguiente forma: un algoritmo de aprendizaje puede tener varios parámetros que permiten cambiar su comportamiento, por ejemplo, se pueden utilizar varios criterios dife-

PEEKING

Figura 18.7 Una curva de aprendizaje para el algoritmo de aprendizaje de árboles de decisión aplicado a 100 ejemplos del dominio del restaurante generados aleatoriamente. La gráfica resume 20 pruebas.

rentes para elegir el siguiente atributo en el aprendizaje del árbol de decisión. Generamos hipótesis para diversos valores de los parámetros, midiendo su proporción de aciertos frente al conjunto de test e informando sobre este valor para la mejor hipótesis. ¡Se ha producido *peeking*! El motivo es que la hipótesis ha sido seleccionada *tomando como base su calidad sobre el conjunto de test*, por lo tanto, se ha filtrado información sobre el conjunto de test en el algoritmo de aprendizaje. La moraleja de todo esto es que cualquier proceso que suponga comparar la calidad de una hipótesis frente a un conjunto de test, debe utilizar un *nuevo* conjunto de test para medir la calidad de la hipótesis que finalmente se haya seleccionado. En la práctica esto es muy complicado, por eso la gente continúa haciendo experimentos sobre conjuntos de test contaminados.

Ruido y sobreajuste

Como ya se ha mostrado, si hay dos o más ejemplos con la misma descripción (en términos de los atributos), pero diferentes clasificaciones, el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN fallará en la búsqueda de un árbol de decisión que sea consistente con todos los ejemplos. La solución que se mencionó anteriormente consiste en tener en cada nodo hoja o bien la clasificación de la mayoría de su conjunto de ejemplos, si se requiere una hipótesis determinística, o bien las probabilidades estimadas de cada clasificación utilizando frecuencias relativas. Desafortunadamente, esto no es todo. Es bastante posible, y de hecho probable, que incluso cuando falta información vital, el algoritmo de aprendizaje de árboles de decisión encuentre un árbol de decisión consistente con todos los ejemplos. Esto pasa porque el algoritmo puede utilizar los atributos irrelevantes, si hay alguno, para hacer distinciones falsas entre los ejemplos.

Considere el problema de intentar predecir el resultado del lanzamiento de un dado. Suponga que se han llevado a cabo experimentos con varios dados durante un largo pe-

ríodo de tiempo y que los atributos que describen cada ejemplo de entrenamiento son los siguientes:

1. *Día*: día en que el dado fue lanzado (Lun, Mar, Mier, Jue).
2. *Mes*: mes en que el dado fue lanzado (Ene o Feb).
3. *Color*: color del dado (Rojo o Azul).

Mientras dos ejemplos no tengan la misma descripción, el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN encontrará una hipótesis exacta. Cuantos más atributos haya, más probable es que se pueda encontrar una hipótesis exacta. Ninguna de estas hipótesis será totalmente falsa. Lo que nos gustaría es que APRENDIZAJE-ÁRBOL-DECISIÓN devolviera un nodo hoja simple con probabilidades cercanas a 1/6 para cada lanzamiento, utilizando suficientes ejemplos de entrenamiento.

Hay que tener cuidado de no utilizar el grado de libertad que aparece cuando hay un conjunto grande de hipótesis posibles, para encontrar «regularidades» poco significativas en los datos. Este problema se denomina **sobreajuste**. El sobreajuste es un fenómeno muy generalizado que ocurre cuando la función principal no es del todo aleatoria. Afecta a todos los tipos de algoritmos de aprendizaje, no sólo a los árboles de decisión.

SOBREAJUSTE

El tratamiento matemático completo del sobreajuste está fuera del ámbito de este libro. Aquí presentamos para afrontar el problema una técnica sencilla denominada **poda del árbol de decisión**. La poda funciona impidiendo divisiones recursivas sobre atributos que no son claramente relevantes, incluso cuando los datos en ese nodo del árbol no estén clasificados de forma uniforme. La pregunta es: ¿cómo se detecta un atributo irrelevante?

PODA DEL ÁRBOL DE DECISIÓN

Suponga que dividimos el conjunto de ejemplos utilizando un atributo irrelevante. Generalmente, esperaremos que los subconjuntos resultantes tengan aproximadamente la misma proporción de cada clase que el conjunto original. En este caso, la ganancia de información será cercana a cero[4]. Así, la ganancia de información es una buena indicación de irrelevancia. Ahora la pregunta es: ¿qué valor mínimo se debe exigir a la ganancia para hacer la división sobre un atributo?

Esta pregunta se puede responder utilizando un **test de significancia** estadístico. El test comienza asumiendo que no existe ningún patrón en los datos (la denominada **hipótesis nula**). Después, los datos actuales son analizados para calcular cuánto se desvían de una ausencia perfecta de patrón. Si el grado de desviación es estadísticamente improbable (normalmente tomando como media un 5 por ciento de probabilidad o menos), entonces se considera que es una buena evidencia de la presencia de un patrón significativo en los datos. Las probabilidades se calculan a partir de distribuciones estándar de la cantidad de desviación que se esperaría en un muestreo aleatorio.

TEST DE SIGNIFICANCIA

HIPÓTESIS NULA

En este caso, la hipótesis nula es que el atributo es irrelevante, y por lo tanto, que la ganancia de información para un ejemplo infinitamente grande sería cero. Necesitamos calcular la probabilidad de que, bajo la hipótesis nula, un ejemplo de tamaño v exhiba la desviación observada a partir de la distribución esperada de ejemplos positivos y negativos. Podemos medir la desviación comparando el número real de ejemplos positi-

[4] De hecho, la ganancia de información será positiva a menos que las proporciones sean todas exactamente la misma. (*Véase* Ejercicio 18.10).

vos y negativos en cada subconjunto, p_i y n_i, con los números esperados asumiendo irrelevancia real, y:

$$\hat{p}_i = p \times \frac{p_i + n_i}{p + n} \qquad \hat{n}_i = n \times \frac{p_i + n_i}{p + n}$$

Una medida conveniente de la desviación total viene dada por

$$D = \sum_{i=1}^{v} \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{\hat{n}_i}$$

Bajo la hipótesis nula, el valor de D se distribuye de acuerdo con la distribución χ^2 (chi-cuadrado) con $v - 1$ grados de libertad. La probabilidad de que el atributo sea realmente irrelevante se puede calcular con ayuda de las tablas de la χ^2 estándar, o con *software* estadístico. El Ejercicio 18.11 propone que se realicen los cambios oportunos en APRENDIZAJE-ÁRBOL-DECISIÓN para implementar esta forma de poda, que se conoce como **poda χ^2.**

PODA χ^2

Con poda, el ruido se puede tolerar: los errores de clasificación proporcionan un incremento lineal en la predicción del error, mientras que los errores en la descripción de los ejemplos tienen un efecto asintótico que es peor a medida que el árbol se va reduciendo a conjuntos pequeños. Cuando los datos contienen mucho ruido, los árboles que se construyen con poda funcionan significativamente mejor que los que se construyen sin poda. Por lo general, los árboles podados son más pequeños y por lo tanto más sencillos de entender.

VALIDACIÓN CRUZADA

La **validación cruzada** (*cross-validation*) es otra técnica que reduce el sobreajuste. Puede ser aplicada a cualquier algoritmo de aprendizaje, no sólo a los árboles de decisión. La idea básica es estimar la calidad de cada hipótesis en la predicción de datos no observados. Esto se hace separando una parte de datos conocidos y utilizándola para medir la calidad de la predicción de una hipótesis inducida a partir de los datos restantes. La *validación cruzada de K pasadas* (*K-fold cross-validation*) consiste en realizar k experimentos, dejando a un lado cada vez $1/k$ de los datos para test y promediando los resultados. Los valores típicos de k son 5 y 10. El extremo es $k = n$, también conocido como *validación cruzada omitiendo uno* (*leave-one-out-cross-validation*). La validación cruzada se puede utilizar en conjunción con cualquier método de construcción del árbol (incluyendo poda) para seleccionar un árbol con una calidad de predicción buena. Para evitar el problema de *peeking*, debemos medir esta calidad con un nuevo conjunto de test.

Extensión de la aplicabilidad de los árboles de decisión

Para extender la inducción de árboles de decisión a una mayor variedad de problemas es necesario tratar algunos aspectos que mencionaremos brevemente. Para un entendimiento completo le sugerimos que realice los ejercicios asociados.

- **Datos perdidos**: en muchos dominios no son conocidos todos los valores de los atributos para cada ejemplo. Los valores pueden haberse perdido, o puede que sea muy caro obtenerlos. Esto da lugar a dos problemas: primero, dado un árbol de decisión completo, ¿cómo se debe clasificar un objeto para el que no se conoce uno de sus atributos de test? Segundo, ¿cómo se debe modificar la fórmula de la

ganancia de información cuando algunos ejemplos tienen valores desconocidos para cierto atributo? Estas cuestiones se tratan en el Ejercicio 18.12.

- **Atributos multivaluados**: cuando un atributo tiene muchos valores posibles, la ganancia de información proporciona una indicación inapropiada de su utilidad. En el caso extremo, se podría utilizar un atributo, como *NombreRestaurante*, que tenga un valor diferente para cada ejemplo. En este caso, cada subconjunto de ejemplos tendrá un único elemento con una única clasificación, por lo tanto la medida de ganancia de información tendrá su máximo valor para este atributo. Sin embargo, el atributo puede ser irrelevante o inútil. Una solución es utilizar el **ratio de ganancia** (Ejercicio 18.13).

RATIO DE GANANCIA

- **Atributos de entrada continuos de valor entero**: los atributos continuos de valor entero, como *Altura* o *Peso*, tienen un número infinito de valores posibles. Más que generar infinitas ramas, los algoritmos de aprendizaje del árbol de decisión típicamente encuentran el **punto de ruptura** (*split point*) que proporciona la máxima ganancia de información. Por ejemplo, en un nodo del árbol se puede dar el caso de que el test sobre *Altura>160* proporcione la mayor información. Existen métodos de programación dinámica eficiente para encontrar puntos de ruptura, pero esta parte es, con diferencia, la más ineficiente de las aplicaciones reales de aprendizaje de árboles de decisión.

PUNTO DE RUPTURA

- **Atributos de salida de valor continuo**: si intentamos predecir un valor numérico, como el precio de una obra de arte, más que una clasificación discreta necesitamos un **árbol de regresión**. Este tipo de árbol tiene en cada hoja una función lineal de algún subconjunto de atributos numéricos, en vez de un valor simple. Por ejemplo, la rama para grabados pintados a mano puede terminar con una función lineal del área, edad y número de colores. El algoritmo de aprendizaje debe decidir cuándo dejar de dividir para comenzar a aplicar regresión lineal utilizando los atributos restantes (o algún subconjunto).

ARBOL DE REGRESIÓN

Un sistema de aprendizaje de árboles de decisión para aplicaciones reales debe ser capaz de manejar todos estos problemas. El manejo de variables de valor continuo es especialmente importante, porque tanto los procesos físicos como los financieros proporcionan datos numéricos. Se han construido algunos paquetes comerciales que utilizan estos criterios, y que han sido utilizados para desarrollar sistemas en muchos campos. En muchas áreas de industria y comercio, los árboles de decisión constituyen normalmente el primer método que se utiliza cuando el objetivo es extraer un método de clasificación del conjunto de datos. Una propiedad importante de los árboles de decisión es que los humanos pueden entender la salida del algoritmo de aprendizaje. (Por supuesto, esto es un *requisito legal* para las decisiones financieras que son objeto de leyes de anti-discriminación). Esta propiedad no la comparten las redes de neuronas (*véase* Capítulo 20).

18.4 Aprendizaje de conjuntos de hipótesis

Hasta ahora hemos estudiado métodos de aprendizaje en los que para hacer predicciones se utiliza una única hipótesis, elegida de un espacio de hipótesis. La idea de los métodos de aprendizaje de conjuntos de hipótesis (***ensemble learning***) es seleccionar una colección, o conjunto, del espacio de hipótesis y combinar sus predicciones. Por ejem-

ENSEMBLE LEARNING

plo, podemos generar cien árboles de decisión diferentes a partir del mismo conjunto de entrenamiento y votar por la mejor clasificación para un nuevo ejemplo.

La motivación del aprendizaje de conjuntos de hipótesis es sencilla. Considere un conjunto de $M = 5$ hipótesis y suponga que combinamos sus predicciones utilizando simplemente el voto de la mayoría. Para que el método de aprendizaje de conjuntos de hipótesis clasifique mal un ejemplo, *como mínimo tres de las cinco hipótesis lo tienen que clasificar mal*. La esperanza es que esto sea menos probable que el hecho de que una única hipótesis clasifique mal. Suponga que asumimos que cada hipótesis h_i de la colección tiene un error de p, es decir, la probabilidad de que un ejemplo elegido aleatoriamente sea mal clasificado por h_i es p. Además, suponga que asumimos que los errores cometidos por cada hipótesis son *independientes*. En este caso, si p es pequeño, la probabilidad de que se produzca un gran número de errores de clasificación es minúscula. Por ejemplo, un sencillo cálculo (Ejercicio 18.14) muestra que utilizando un conjunto de cinco hipótesis se reduce la tasa de error de 1 sobre 10, a una tasa de error más pequeña que 1 sobre 100. Si bien, es obvio que la asunción de independencia no es razonable, porque es probable que las hipótesis sean engañosas de la misma forma que lo son algunos aspectos de los datos de entrenamiento. Pero si las hipótesis son al menos un poco diferentes, reduciendo así la correlación entre sus errores, el aprendizaje de conjuntos de hipótesis puede ser muy útil.

También se puede pensar en la idea de conjunto de hipótesis como una forma genérica de ampliar el espacio de hipótesis. Es decir, asumir que el conjunto de hipótesis propiamente dicho es una hipótesis y el nuevo espacio de hipótesis es el conjunto de todos los posibles conjuntos que se pueden construir a partir de las hipótesis del espacio original. La Figura 18.8 muestra de qué forma esto puede dar lugar a un espacio de hipótesis más expresivo. Si un algoritmo de aprendizaje que genera el conjunto de hipótesis original es simple y eficiente, el método de conjuntos de hipótesis proporciona una for-

Figura 18.8 Ilustración del incremento de la potencia expresiva obtenida con aprendizaje de conjuntos de hipótesis. Tomamos tres hipótesis lineales, cada una de las cuales clasifica positivamente en la zona no sombreada, y se clasifica como positivo cualquier ejemplo clasificado positivamente por las tres. La región triangular resultante es una hipótesis que no se puede expresar en el espacio de hipótesis original.

ma de aprender un tipo de hipótesis mucho más expresivas, sin incurrir demasiado en un aumento de la complejidad computacional o algorítmica.

BOOSTING

CONJUNTO DE ENTRENAMIENTO CON PESOS

El método de conjuntos de hipótesis más comúnmente utilizado es el denominado ***boosting*** *(propulsión)*. Para entender cómo funciona, es necesario explicar primero la idea de **conjunto de entrenamiento con pesos** (***weighted training set***). En este tipo de conjunto de entrenamiento cada ejemplo lleva asociado un peso $w_j \geq 0$. Cuanto mayor sea el peso de un ejemplo, más importancia tiene éste durante el aprendizaje de una hipótesis. Es sencillo modificar los algoritmos de aprendizaje que hemos visto hasta ahora para que operen con conjuntos de entrenamiento con pesos[5].

El *boosting* comienza con $w_j = 1$ para todos los ejemplos (es decir, con un conjunto de entrenamiento normal). A partir de este conjunto, genera la primera hipótesis, h_1. Esta hipótesis clasificará correctamente algunos de los ejemplos de entrenamiento, e incorrectamente otros. Nos gustaría que la siguiente hipótesis realizara un mejor trabajo con los ejemplos mal clasificados, para ello incrementamos sus pesos y decrementamos los pesos de los ejemplos que han sido correctamente clasificados. A partir de este conjunto de entrenamiento con nuevos pesos, generamos la hipótesis h_2. El proceso continúa de esta forma hasta que se han generado M hipótesis, donde M es una entrada del algoritmo de *boosting*. El conjunto final de hipótesis es una combinación ponderada de todas las hipótesis de M, donde el peso de cada una es una función de lo bien que clasifica el conjunto de entrenamiento. La Figura 18.9 muestra el funcionamiento del algoritmo

Figura 18.9 Funcionamiento del algoritmo de *boosting*. Cada rectángulo sombreado corresponde a un ejemplo; la altura del rectángulo equivale al peso. Los símbolos √ y × indican si el ejemplo ha sido clasificado correctamente o no por la hipótesis actual. El tamaño del árbol de decisión indica el peso de esta hipótesis en la colección final de hipótesis.

[5] Para algoritmos de aprendizaje en los que esto no es posible, se puede crear una **réplica del conjunto de entrenamiento** donde el ejemplo i-ésimo aparece w_j veces, utilizando aleatoriedad para manejar pesos fraccionarios.

desde un punto de vista conceptual. Existen muchas variantes de la idea básica de *boosting,* que utilizan diferentes formas de ajustar los pesos y combinar las hipótesis. En la Figura 18.10 se muestra un algoritmo específico denominado ADABOOST. Aunque los detalles sobre el ajuste de pesos no son muy importantes, ADABOOST tiene una propiedad muy importante: si el algoritmo de aprendizaje de entrada L es un algoritmo de **aprendizaje débil,** lo que significa que L siempre devuelve una hipótesis con un error sobre el conjunto de entrenamiento que es ligeramente mejor que una suposición aleatoria (es decir, 50 por ciento para clasificación booleana), el ADABOOST devolverá una hipótesis que *clasifica los datos de entrenamiento perfectamente* para M suficientemente grande. Así, el algoritmo aumenta la precisión sobre los datos de entrenamiento del algoritmo original. Este resultado se mantiene siempre, sin verse afectado por la inexpresividad del espacio de hipótesis original ni por la complejidad de la función que esté siendo aprendida.

APRENDIZAJE DÉBIL

Veamos cómo funciona el *boosting* sobre los datos del restaurante. Elegiremos como espacio de hipótesis original el conjunto de los árboles de decisión con exactamente un test en la raíz (***decision stumps***). La curva que aparece por debajo en la Figura 18.11(a) muestra que sin utilizar *boosting*, estos árboles no son muy efectivos para este conjunto de datos, alcanzándose un acierto de la predicción de sólo el 81 por ciento de 100 ejemplos de entrenamiento. Cuando se aplica *boosting* (con $M = 5$), la proporción de aciertos mejora, llegando al 93 por ciento después de 100 ejemplos.

DECISION STUMP

El incremento del tamaño del conjunto M, da lugar a algo interesante. La Figura 18.11(b) muestra la precisión del conjunto de entrenamiento (sobre 100 ejemplos) como una función de M. Nótese que el error llega a ser cero (como dice el teorema de *boosting*) cuando M es 20; es decir, una combinación ponderada de 20 *decision stumps*

```
función ADABOOST (ejemplos, L, M) devuelve hipótesis ponderada
  entrada: ejemplos, conjunto de N ejemplos etiquetados (x1, y1),...,(xN, yN)
           L, un algoritmo de aprendizaje
           M, número de hipótesis en el conjunto de hipótesis
  variables locales: w, un vector de N pesos para los ejemplos, inicialmente cada uno es1/N
                     h, un vector de M hipótesis
                     z, un vector de M pesos para las hipótesis

  para m = 1 a M hacer
     h[m] ← L(ejemplos,w)
     error ← 0
     para j = 1 a N hacer
        si h[m](xj) ≠ yj entonces error ← error + w[j]
     para j = 1 a N hacer
        si h[m](xj) = yj entonces w[j] ← w[j] · error / (1 − error)
     w ← NORMALIZAR(w)
     z[m] ← log(1-error) / error
  devolver MAYORÍA-PONDERADA(h, z)
```

Figura 18.10 La variante ADABOOST del método de *boosting* para aprendizaje de conjuntos de hipótesis. El algoritmo combina hipótesis.

La función MAYORÍA-PONDERADA genera una hipótesis que devuelve el valor de salida con el voto más alto, a partir de las hipótesis en **h**, y con los pesos para cada voto indicados por **z**.

Figura 18.11 (a) Gráfico que muestra el rendimiento de los *decision stumps* con *boosting* para $M = 5$ y el ejemplo del restaurante. (b) Proporción correcta en el conjunto de entrenamiento y en el de test, como una función de M, el número de hipótesis en el conjunto de hipótesis. Observe que la precisión del conjunto de test mejora ligeramente incluso después de que la precisión del conjunto de test alcance 1, es decir, después de que el conjunto de hipótesis encaje exactamente con los datos.

es suficiente para clasificar exactamente los 100 ejemplos. Si se añaden más *decision stumps* al conjunto, el error permanece en cero. La gráfica también muestra que la *precisión del conjunto de test continúa incrementándose después de que el error en el conjunto de entrenamiento haya llegado a cero*. Con $M = 20$, la precisión del test es 0,95 (o 0,05 de error), y aumenta a 0,98 cuando $M = 137$, antes de bajar gradualmente a 0,95.

Este hallazgo, que es bastante robusto a través de conjuntos de datos y espacios de hipótesis, fue muy sorprendente cuando se observó por primera vez. La *navaja de Ockham* indica que es mejor no hacer hipótesis más complejas de lo necesario, sin embargo, la gráfica indica que la predicción mejora a medida que el conjunto de hipótesis se hace más complejo. Se han propuesto algunas explicaciones de esto. Una explicación es que el *boosting* aproxima el **aprendizaje Bayesiano** (*véase* Capítulo 20), del que se puede demostrar que es un algoritmo de aprendizaje óptimo, y la aproximación mejora a medida que se añaden más hipótesis. Otra explicación posible es que la adición de más hipótesis permite que el conjunto de hipótesis discrimine más claramente entre ejemplos positivos y negativos, lo cual ayuda a la hora de clasificar nuevos ejemplos.

18.5 ¿Por qué funciona el aprendizaje?: teoría computacional del aprendizaje

En el Apartado 18.2 quedó pendiente la siguiente cuestión: ¿cómo podemos estar seguros de que la teoría que ha generado el algoritmo de aprendizaje predecirá correctamente el futuro? En términos formales, ¿cómo sabremos que la hipótesis h está cercana a la

función objetivo f si no conocemos f? Estas preguntas han sido meditadas durante siglos. Hasta que se encuentre respuesta para ellas, el aprendizaje automático estará preguntándose por sus propios éxitos.

TEORÍA COMPUTACIONAL DEL APRENDIZAJE

El enfoque tomado en esta sección se basa en la **teoría computacional del aprendizaje**, un campo entre la IA, la estadística y la informática teórica. El principio fundamental es el siguiente: *cualquier hipótesis que sea muy errónea será descubierta con una probabilidad alta después de un número pequeño de ejemplos, ya que realizará una predicción incorrecta. Por ello, es improbable que cualquier hipótesis que es consistente con un conjunto suficientemente grande de ejemplos de entrenamiento, sea muy errónea: es decir, debe ser una* ***aproximación correcta probable (PAC)***. Cualquier algoritmo de aprendizaje que devuelve hipótesis que sean una aproximación correcta probable se denomina algoritmo de **PAC-aprendizaje** (*PAC-learning*).

APROXIMACIÓN CORRECTA PROBABLE

PAC-APRENDIZAJE

En el argumento anterior existen algunas sutilezas. La principal cuestión es la conexión entre el conjunto de ejemplos de entrenamiento y de test; después de todo, queremos que la hipótesis sea aproximadamente correcta en el conjunto de test, no sólo en el conjunto de entrenamiento. La suposición clave es que los conjuntos de ejemplos de entrenamiento y de test son elegidos de forma fortuita e independiente a partir del mismo conjunto de ejemplos que siguen una misma distribución de probabilidad. Esto se denomina suposición **estacionaria**. Sin la suposición estacionaria, la teoría no podría hacer ninguna predicción acerca del futuro, porque no existiría la conexión necesaria entre el futuro y el pasado. La suposición estacionaria equivale a suponer que el proceso de selección de ejemplos no es malévolo. Obviamente, si el conjunto de entrenamiento está formado sólo por ejemplos extraños (perros con dos cabezas, por ejemplo) el algoritmo de aprendizaje no podrá ayudar, ya que hará generalizaciones sin éxito para reconocer perros.

ESTACIONARIA

¿Cuántos ejemplos se necesitan?

Para poner en práctica lo anterior, necesitaremos algunas notaciones:

- Sea **X** el conjunto de todos los posibles ejemplos.
- Sea D la distribución, a partir de la cual los ejemplos son escogidos.
- Sea **H** el conjunto de las hipótesis posibles.
- Sea N el número de ejemplos en el conjunto de entrenamiento.

Inicialmente, asumiremos que la función verdadera f pertenece a **H**. Ahora, podemos definir el **error** de una hipótesis h con respecto a la función verdadera f, dada una distribución D sobre los ejemplos, como la probabilidad que h sea diferente de f con respecto a un ejemplo:

ERROR

$$\text{error}(h) = \text{P}(h(x) \neq f(x) | x \text{ se obtiene a partir de } D)$$

Esta es la misma cantidad medida experimentalmente por las curvas de aprendizaje que se mostraron anteriormente.

Una hipótesis h se denomina **aproximadamente correcta** si $\text{error}(h) \leq \epsilon$, donde ϵ es una constante pequeña. Pretendemos mostrar que después de observar N ejemplos, con probabilidad alta, todas las hipótesis consistentes serán aproximadamente correctas. Se puede pensar que una hipótesis aproximadamente correcta estará «cerca» de la

ϵ-BOLA

función verdadera en el espacio de hipótesis: caerá dentro de lo que se denomina la **ϵ-bola** alrededor de la función verdadera f. La Figura 18.12 muestra el conjunto de todas las hipótesis **H**, dividido en la zona que cubre la ϵ-bola alrededor de f y el resto, que denominaremos $\mathbf{H}_{\text{mala}}$.

A continuación se muestra cómo podemos calcular la probabilidad de que una hipótesis «seriamente errónea» $h_b \in \mathbf{H}_{\text{mala}}$ sea consistente con los N primeros ejemplos. Sabemos que error$(h_b) > \epsilon$. Por ello, la probabilidad de que se ajuste a un ejemplo dado es de al menos $1 - \epsilon$. El límite para N ejemplos es

$$P(h_b \text{ se ajuste a } N \text{ ejemplos}) \leq (1 - \epsilon)^N$$

La probabilidad de que $\mathbf{H}_{\text{mala}}$ contenga al menos una hipótesis consistente está limitada por la suma de las probabilidades individuales:

$$\text{P}(\mathbf{H}_{\text{mala}} \text{ contenga una hipótesis consistente}) \leq |\mathbf{H}_{\text{mala}}|(1 - \epsilon)^N \leq |\mathbf{H}|(1 - \epsilon)^N ,$$

donde se ha tenido en cuenta el hecho de que $|\mathbf{H}_{\text{mala}}| \leq |\mathbf{H}|$. Buscamos que la probabilidad de que esto suceda quede por debajo de un número pequeño δ:

$$|\mathbf{H}|(1 - \epsilon)^N \leq \delta$$

Dado que $1 - \epsilon \leq e^{-\epsilon}$, podemos conseguirlo si entrenamos el algoritmo con el siguiente número de ejemplos:

$$N \geq \frac{1}{\epsilon}\left(\ln \frac{1}{\delta} + \ln |\mathbf{H}|\right) \tag{18.1}$$

por ello, si un algoritmo de aprendizaje devuelve una hipótesis que es consistente con bastantes ejemplos, con probabilidad de al menos $1 - \delta$, comete como máximo un error ϵ. En otras palabras, es una aproximación correcta probable. El número de ejemplos que se requieren, en función de ϵ y δ, se denomina **complejidad de la muestra** del espacio de hipótesis.

COMPLEJIDAD DE LA MUESTRA

Figura 18.12 Diagrama esquemático del espacio de hipótesis, mostrando la zona que cubre la ϵ-bola alrededor de la función f.

Esto demuestra que la cuestión clave es el tamaño del espacio de hipótesis. Como se mostró anteriormente, si **H** es el conjunto de todas las funciones Booleanas sobre n atributos, entonces $|\mathbf{H}| = 2^{2^n}$. De este modo, la complejidad de la muestra del espacio aumenta hasta 2^n. Ya que el número de posibles ejemplos es también 2^n, cualquier algoritmo de aprendizaje para el espacio de todas las funciones Booleanas, que devuelva simplemente una hipótesis consistente con todos los ejemplos conocidos, no se comportará mejor que una tabla de búsqueda (*lookup table*). Otra forma de verlo es observar que para cualquier ejemplo con el que no se haya entrenado, serán tan consistentes aquellas hipótesis del espacio de hipótesis que lo clasifican como positivo, como las que lo clasifican como negativo.

El dilema al que nos enfrentamos es que, a menos que restrinjamos el espacio de funciones que el algoritmo considera, éste no va a ser capaz de aprender; pero si restringimos el espacio, podríamos eliminar la función verdadera. Existen dos modos de «escaparnos» de este dilema. El primer modo es exigir que el algoritmo devuelva no sólo cualquier hipótesis consistente, sino preferiblemente la más sencilla (como se hacía en el aprendizaje de árboles de decisión). El análisis teórico de estos algoritmos está fuera del alcance de este libro, pero en los casos en los que es tratable encontrar hipótesis consistentes sencillas, los resultados de la complejidad de la muestra son generalmente mejores que el análisis basado sólo en la consistencia. El segundo modo, que se muestra aquí, es centrarnos en el aprendizaje de subconjuntos del conjunto total de las funciones Booleanas. La idea es que en la mayoría de los casos no necesitamos el poder total de expresividad de las funciones Booleanas, y podemos trabajar con lenguajes más restrictivos. A continuación, examinamos con más detalle uno de estos lenguajes restringidos.

Aprendizaje de listas de decisión

LISTA DE DECISIÓN

Una **lista de decisión** es una expresión lógica de un formulario restringido. Está formada por una serie de test, cada uno de los cuales es una conjunción de literales. Si el test tiene éxito cuando se aplica a una descripción de un ejemplo, la lista de decisión especifica el valor a ser devuelto. Si el test falla, el proceso continúa con el siguiente test de la lista[6]. Las listas de decisión se parecen a los árboles de decisión, pero su estructura completa es más sencilla. En contraste, los test individuales son más complejos. La Figura 18.13 muestra una lista de decisión que representa la siguiente hipótesis:

$$\forall x\ Esperar(x) \Leftrightarrow Clientes(x, Algunos) \vee (Clientes(x, Lleno) \wedge Vier/Sab(x))$$

Figura 18.13 Una lista de decisión para el problema del restaurante.

[6] Una lista de decisión es por ello idéntica en estructura a una sentencia COND en Lisp.

Si permitimos test de tamaño arbitrario, las listas de decisión pueden representar cualquier función Booleana (Ejercicio 18.15). Por otro lado, si restringimos el tamaño de cada test a como máximo k literales, es posible para el algoritmo de aprendizaje generalizar con éxito a partir de un número pequeño de ejemplos. A este lenguaje le denominamos k-**DL**. El ejemplo de la Figura 18.13 está en 2-DL. Es fácil mostrar (Ejercicio 18.15) que k-**DL** incluye como un subconjunto el lenguaje k-**DT**, el conjunto de todos los árboles de decisión de profundidad máxima k. Es importante recordar que el lenguaje referenciado por k-**DL** depende de los atributos que se han utilizado para describir los ejemplos. Usaremos la notación k-**DL**(n) para denotar a los lenguajes k-**DL** que utilizan n atributos Booleanos.

***K*-DL**

***K*-DT**

La primera tarea es mostrar que k-**DL** se puede aprender, es decir, que cualquier función en k-**DL** puede ser aproximada con precisión después del entrenamiento con un número razonable de ejemplos. Para hacer esto, necesitamos calcular el número de hipótesis en el lenguaje. Sea $Conj(n,k)$ el lenguaje de test, conjunciones de como máximo k literales usando n atributos. Ya que una lista de decisión se construye a partir de test, y dado que a cada test se le puede asignar un resultado de *Sí* o *No,* o puede estar fuera de la lista de decisión, existen como mucho $3^{|Conj(n,k)|}$ conjuntos distintos de test componentes. Cada uno de estos conjuntos de test pueden estar en cualquier orden, así que

$$|k\text{-DL}(n)| \leq 3^{|Conj(n,k)|}|Conj(n,k)|!$$

El número de conjunciones de k literales a partir de n atributos viene dado por

$$|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k)$$

Por lo tanto, después de algunos cálculos, obtenemos

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}$$

Podemos introducirlo en la Ecuación (18.1) para mostrar que el número de ejemplos que se necesitan para el PAC-aprendizaje de una función k-DL es polinómico en n:

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right)$$

Por ello, cualquier algoritmo que devuelva una lista de decisión consistente haciendo uso del PAC-aprendizaje aprenderá una función k-DL con un número razonable de ejemplos, para k pequeñas.

La siguiente tarea es encontrar un algoritmo eficiente que devuelva una lista de decisión consistente. Utilizaremos un algoritmo voraz denominado APRENDIZAJE-LISTA-DECISIÓN, que repetidas veces encuentra un test que verifica con exactitud algún subconjunto del conjunto de entrenamiento. Una vez que se encuentra dicho test, lo añade a la lista de decisión que se está construyendo y elimina los correspondientes ejemplos. Luego, construye el resto de la lista de decisión, haciendo uso sólo de los ejemplos restantes. Esto se repite hasta que no hay ejemplos. El algoritmo se muestra en la Figura 18.14.

función Aprendizaje-Lista-Decisión (*ejemplos*) **devuelve** una lista de decisión, o *fallo*

si *ejemplos* esta vacía **entonces devolver** la lista de decisión trivial *No*
$t \leftarrow$ un test que encaje un subconjunto no vacío de $ejemplos_t$ de *ejemplos* de forma que los elementos de $ejemplos_t$ sean todos positivos o todos negativos
si no se puede formar t **entonces** *fallo*
si los ejemplos en $ejemplos_t$ son positivos **entonces** $r \leftarrow$ *Sí* **si no** $r \leftarrow$ *No*
devolver la lista de decisión con el test inicial t y resultado r y los restantes test dados por APRENDIZAJE-LISTA-DECISIÓN ($ejemplos$-$ejemplos_t$)

Figura 18.14 Un algoritmo de aprendizaje de listas de decisión.

Figura 18.15 Curva de aprendizaje para el algoritmo APRENDIZAJE-LISTA-DECISIÓN sobre los datos del restaurante. La curva del algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN se muestra para comparar.

Este algoritmo no especifica el método para elegir el siguiente test que se añade a la lista de decisión. Aunque los resultados formales proporcionados anteriormente no dependen del método de selección, sería razonable preferir test pequeños que encajen conjuntos grandes de ejemplos uniformemente clasificados, para que la lista de decisión total sea lo más compacta posible. La estrategia más sencilla consiste en encontrar el test más pequeño t que encaje uniformemente cualquier subconjunto clasificado, independientemente del tamaño del subconjunto. Este enfoque funciona bastante bien, como muestra la Figura 18.15.

Discusión

IDENTIFICAR EN EL LÍMITE

La teoría computacional del aprendizaje ha generado una nueva forma para considerar el problema de aprendizaje. En los inicios de la década de los 60 la teoría del aprendizaje se centraba en el problema de **identificar en el límite**. De acuerdo con esta noción, un algoritmo de identificación debe devolver una hipótesis que encaje con exactitud con la función real. Una forma de realizarlo se muestra a continuación: primero, se ordenan

todas las hipótesis en **H** de acuerdo a cierta medida de simplicidad. Luego, se elige la hipótesis más sencilla, consistente con todos los ejemplos hasta el momento. A medida que aparecen nuevos ejemplos, el método abandonará una hipótesis más sencilla que se invalida y adoptará, en lugar de ella, una más compleja. Una vez que se alcanza la función real, nunca se abandonará. Desgraciadamente, en muchos espacios de hipótesis, el número de ejemplos y el tiempo de computación requerido para alcanzar la función real son enormes. Por ello, la teoría computacional del aprendizaje no insiste en que el agente de aprendizaje encuentre «una ley verdadera» que gobierne su entorno, sino en que encuentre una hipótesis con un cierto grado de precisión en la predicción. La teoría computacional del aprendizaje también trata el compromiso entre la expresividad del lenguaje de la hipótesis y la complejidad del aprendizaje y ha llevado directamente a una clase importante de algoritmos de aprendizaje denominados máquinas vectoriales soporte (*support vector machines*).

Los resultados del PAC-aprendizaje que se han mostrado, son resultados para el peor caso en complejidad y no reflejan necesariamente el caso medio de la complejidad de la muestra, como las medidas mostradas anteriormente en las curvas de aprendizaje. El análisis de un caso medio debe también hacer suposiciones sobre la distribución de los ejemplos y sobre la distribución de la función real que el algoritmo tiene que aprender. Mientras estos aspectos se comprenden mejor, la teoría computacional del aprendizaje continúa proporcionando valiosos consejos a los investigadores del campo del aprendizaje automático que están interesados en predecir y modificar la habilidad de aprendizaje de sus algoritmos. Además de las listas de decisión, se han obtenido resultados para casi todas las subclases de funciones booleanas conocidas, para redes neuronales (*véase* Capítulo 20), y para conjuntos de sentencias de lógica de primer orden (*véase* Capítulo 19). Los resultados muestran que el problema de aprendizaje inductivo puro, donde el agente comienza sin conocimiento *a priori* sobre la función objetivo, es generalmente muy difícil. Como veremos en el Capítulo 19, el uso de conocimiento a priori para guiar el aprendizaje inductivo hace posible el aprendizaje de conjuntos de sentencias bastante grandes a partir de un número razonable de ejemplos, incluso en un lenguaje tan expresivo como la lógica de primer orden.

18.6 Resumen

En este capítulo nos hemos concentrado en el aprendizaje inductivo de funciones determinísticas a partir de ejemplos. Los principales puntos fueron los siguientes:

- El aprendizaje puede tomar muchas formas, dependiendo de la naturaleza de las herramientas de desarrollo, el componente a ser mejorado, y de la realimentación disponible.
- Si la realimentación disponible, tanto de un profesor como del entorno, proporciona el valor correcto para los ejemplos, el problema de aprendizaje se denomina **aprendizaje supervisado**. La tarea, también llamada **aprendizaje inductivo**, consiste en aprender una función a partir de ejemplos de sus entradas y salidas. El

aprendizaje de una función de valores discretos se denomina **clasificación**; el aprendizaje de una función continua se denomina **regresión**.

- El aprendizaje inductivo consiste en encontrar una hipótesis **consistente** que verifique los ejemplos. La **navaja de Ockham** sugiere elegir la hipótesis consistente más sencilla. La dificultad de esta tarea depende de la representación elegida.
- Los árboles de decisión pueden representar todas las funciones booleanas. La heurística de la **ganancia de información** proporciona un método eficiente para encontrar un árbol de decisión simple y consistente.
- El rendimiento de un algoritmo de aprendizaje se mide a través de la **curva de aprendizaje**, que muestra la precisión de predicción en el conjunto de ejemplos de test como una función del tamaño del conjunto de ejemplos de entrenamiento.
- Los métodos de conjuntos de hipótesis, como el ***boosting***, a menudo se comportan mejor que los métodos individuales.
- La **teoría computacional del aprendizaje** analiza la complejidad de la muestra y la complejidad computacional del aprendizaje inductivo. Existe un compromiso entre la expresividad del lenguaje de la hipótesis y de la facilidad del aprendizaje.

Notas bibliográficas e históricas

En el Capítulo 1 se dio una idea general acerca de la historia de las investigaciones filosóficas dentro del aprendizaje inductivo. A William de Ockham (1280-1349), el filósofo más influyente de su siglo y un gran contribuidor a la epistemología medieval, la lógica y la metafísica, se le atribuye la afirmación denominada «La navaja de Ockham», en latín, *Entia non sunt multiplicanda praeter necessitatem*, y en castellano, «Las entidades no han de ser multiplicadas más allá de la necesidad». Desgraciadamente, este loable consejo no se encuentra en ninguna parte de sus escrituras con estas palabras precisamente.

EPAM («*Elementary Perceiver And Memorizer*»), el memorizador y perceptor elemental (Feigenbaum, 1961), fue uno de los primeros sistemas que utilizaron árboles de decisión (o **redes de discriminación**). EPAM fue diseñado como un modelo de simulación cognitiva de aprendizaje de conceptos humanos. CLS (Hunt *et al*., 1966) usaba un método heurístico para construir árboles de decisión. ID3 (Quinlan, 1979) añadió la idea crucial de usar el contenido de la información para proporcionar la función heurística. La teoría de la información, por su parte, fue desarrollada por Claude Shannon para ayudar en el estudio de la comunicación (Shannon y Weaver, 1949). (Shannon también contribuyó con uno de los primeros ejemplos de aprendizaje automático, un ratón mecánico, llamado Theseus, que aprendía a navegar por tanteo a través de un laberinto.) El método de la χ^2 para poda de árboles, fue descrito por Quinlan (1986). C4.5, un paquete industrial de árboles de decisión, puede encontrarse en Quinlan (1993). En la literatura estadística existe una tradición independiente en el aprendizaje de árboles de decisión. La principal referencia es el libro *Classification and Regression Trees (CART)* (*Árboles de Clasificación y Regresión)* (Breiman *et al*., 1984), conocido como el «libro CART».

Se han probado muchos otros enfoques algorítmicos para el aprendizaje. El enfoque de la **mejor hipótesis actual** mantiene una única hipótesis, especializándola cuando se comprueba que es demasiado general y generalizándola cuando se comprueba que es demasiado específica. Esta es una vieja idea de la filosofía (Mill, 1843). Los primeros trabajos en la filosofía cognitiva también sugerían que es una forma natural de aprendizaje de conceptos en humanos (Bruner *et al*., 1957). En IA, el enfoque está más estrechamente ligado al trabajo de Patrick Winston, cuya tesis doctoral (Winston, 1970) aborda el problema de aprendizaje de descripciones de objetos complejos. El método de **espacio de versiones** (*version space*) (Mitchell, 1977, 1982) toma un enfoque distinto, mantiene el conjunto de *todas* las hipótesis consistentes y elimina aquellas que resultan inconsistentes con nuevos ejemplos. El enfoque fue usado en el sistema experto para química META-DENDRAL (Buchanan y Mitchell, 1978), y más tarde en el sistema LEX de Mitchell (1983), que aprendía a resolver problemas de cálculo. Un tercer hilo de influencia surgió por el trabajo de Michalski y colegas en las series de algoritmos AQ, que aprendían conjuntos de reglas lógicas (Michalski, 1969; Michalski *et al*., 1986b).

BAGGING

El aprendizaje de conjuntos de hipótesis es una técnica cada vez más popular para mejorar el funcionamiento de los algoritmos de aprendizaje. ***Bagging*** (Breiman, 1996), el primer método efectivo, combina hipótesis aprendidas a partir de múltiples conjuntos de datos (*bootstrap*), cada uno generado muestreando el conjunto original de datos. El método ***boosting*,** descrito en este capítulo, se originó con los trabajos teóricos de Schapire (1990). El algoritmo ADABOOST fue desarrollado por Freund y Schapire (1996) y fue analizado teóricamente por Schapire (1999). Friedman *et al*. (2000) explica el *boosting* desde un punto de vista estadístico.

El análisis teórico de los algoritmos de aprendizaje comenzó con el trabajo de Gold (1967) sobre **identificación en el límite**. Este enfoque fue motivado en parte por los modelos de descubrimientos científicos de la ciencia de la filosofía (Popper, 1962), pero se ha aplicado principalmente al problema de aprendizaje de gramáticas a partir de ejemplos de sentencias (Osherson *et al*., 1986).

COMPLEJIDAD DE KOLMOGOROV

Mientras que el enfoque de la identificación en el límite se concentra en la convergencia final, el estudio de la **complejidad de Kolmogorov** o la **complejidad algorítmica**, desarrollados independientemente por Solomonoff (1964) y Kolmogorov (1965), intenta proporcionar una definición formal para la noción de simplicidad que se utiliza en la navaja de Ockham. Para evitar el problema de que la simplicidad dependa de la forma en que está representada la información, se propone que la simplicidad se mida a través de la longitud del programa más corto de una máquina de Turing universal que reproduzca correctamente los datos observados. Aunque existen muchas posibles máquinas de Turing universales, y por lo tanto muchos programas posibles que sean «el más corto», difieren en longitud como mucho en una constante que es independiente de la cantidad de datos. Esta curiosa apreciación, que muestra esencialmente que cualquier sesgo en la representación inicial será finalmente salvado por los propios datos, se viene abajo por la indecibilidad de computar la longitud del programa más corto. Se pueden usar medidas aproximadas, como el **tamaño de la mínima descripción** o MDL (Rissanen, 1984), que han producido excelentes resultados en la práctica. El texto de Li y Vitanyi es la mejor fuente para la complejidad de Kolmogorov.

TAMAÑO DE LA MÍNIMA DESCRIPCIÓN

La teoría computacional del aprendizaje (es decir, la teoría del aprendizaje PAC) fue inaugurada por Leslie Valiant (1984). El trabajo de Valiant enfatiza la importancia de la complejidad computacional y de la muestra. Con Michael Kearns (1990), Valiant mostró que a las clases con varios conceptos no se les puede aplicar el PAC-aprendizaje, aunque esté disponible suficiente información en los ejemplos. Se obtuvieron algunos resultados positivos para listas de decisión (Rivest, 1987).

TEORÍA DE LA CONVERGENCIA UNIFORME

VC DIMENSIÓN

En estadística, se ha desarrollado un enfoque independiente sobre el análisis de la complejidad de la muestra, comenzando con el trabajo de la **teoría de la convergencia uniforme** (Vapnik y Chervonenkis, 1971). El llamado **VC dimensión** proporciona una medida similar, pero más general que la medida $\ln|\mathbf{H}|$, obtenida por el análisis del PAC. La VC dimensión puede aplicarse a las clases de funciones continuas, a las cuales no se les puede aplicar el análisis del PAC. La teoría del PAC-aprendizaje y la teoría del VC fueron inicialmente relacionadas por los «cuatro Alemanes» (ninguno de los cuales realmente es alemán): Blumer, Ehrenfeucht, Haussler y Warmuth (1989). Desarrollos sucesivos en la teoría de VC llevaron a la invención de las **máquinas vectoriales de apoyo** (*support vector machine* o SVM) (Boser *et al*., 1992; Vapnik, 1998), que se describen en el Capítulo 20.

En *Readings in Machine Learning* (Shavlik y Dietterich, 1990) se ha recopilado un gran número de artículos sobre aprendizaje automático. Los dos volúmenes *Machine Learning 1* (Michalski *et al*., 1983) y *Machine Learning 2* (Michalski *et al*., 1986a) también contienen bastantes artículos importantes, al igual que una enorme bibliografía. Weiss y Kulikowski (1991) proporcionan una amplia introducción a los métodos de aprendizaje de funciones en las áreas de aprendizaje automático, estadística y redes neuronales. El proyecto STATLOG (Michie *et al*., 1994) es sin duda la investigación más exhaustiva en la comparación del rendimiento de los algoritmos de aprendizaje. Una buena parte de la investigación actual en el campo del aprendizaje automático, se publica anualmente en las actas de las conferencias: *International Conference on Machine Learning* y *Neural Information Processing System*, en *Machine Learning,* en la revista *Journal of Machine Learning Research*, y en las principales revistas de IA. En los talleres anuales de ACM sobre *Computational Learning Theory* (COLT) también aparecen trabajos sobre la teoría computacional del aprendizaje, que se describe en los textos de Kearns y Vazirani (1994) y Anthony y Bartlett (1999).

EJERCICIOS

18.1 Considere el problema al que se enfrenta un niño pequeño cuando tiene que hablar y entender un lenguaje. Explique cómo este problema encaja dentro del modelo general de aprendizaje, identificando cada uno de los componentes apropiados del modelo.

18.2 Repita el Ejercicio 18.1 para el caso de aprender a jugar al tenis (u otro deporte que le sea familiar). ¿Es aprendizaje supervisado o aprendizaje por refuerzo?

18.3 Dibuje un árbol de decisión para el problema de decidir cuándo atravesar un cruce de carreteras, dado que la luz se acaba de poner verde.

18.4 Nunca se comprueba el mismo atributo dos veces en un mismo camino de un árbol de decisión. ¿Por qué?

18.5 Suponga que generamos un conjunto de entrenamiento a partir de un árbol de decisión y luego aplicamos aprendizaje con árboles de decisión a dicho conjunto. ¿Devolverá el algoritmo de aprendizaje el árbol correcto cuando el tamaño del conjunto de entrenamiento tienda a infinito? ¿Por qué o por qué no?

18.6 Un buen algoritmo de aprendizaje *espantapájaros* es de la siguiente manera: crea una tabla separada con los ejemplos de entrenamiento. Identifica qué salida se produce más veces en los ejemplos de entrenamiento; y la denomina d. Luego, cuando se recibe una entrada que no está en la tabla, se devuelve d. Para entradas que sí están en la tabla, se devuelve la salida que tienen asociada (o la salida más frecuente, si existe más de una). Implemente este algoritmo y determine en qué medida funciona bien en el dominio del restaurante. Esto le debería dar una idea de la línea base del dominio (se debería obtener como mínimo el rendimiento que con cualquier otro algoritmo).

18.7 Suponga que está ejecutando un nuevo algoritmo para un experimento de aprendizaje. Tiene un conjunto de datos formado por 25 ejemplos, para cada uno de dos clases. Decida usar validación cruzada omitiendo uno (*leave-one-out-cross-validation*). Para empezar, ejecute su experimento con un clasificador mayoritario simple. (Un clasificador mayoritario recibe un conjunto de datos de entrenamiento y luego devuelve siempre la clase a la cual pertenecen la mayoría del conjunto de entrenamiento, a pesar de la entrada). Se esperará que el clasificador mayoritario obtenga una puntuación de un 50 por ciento en validación cruzada omitiendo uno, sin embargo para su sorpresa, obtiene puntuación cero. ¿Puede explicar por qué?

18.8 En la construcción recursiva de árboles de decisión, a veces ocurre que en un nodo hoja queda un conjunto formado por ejemplos positivos y negativos, incluso después de que todos los atributos hayan sido evaluados. Suponga que tenemos p ejemplos positivos y n ejemplos negativos.

a) Compruebe cómo la solución que usa el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN, que selecciona la clasificación mayoritaria, minimiza el valor absoluto del error sobre el conjunto de ejemplos en el nodo hoja.

PROBABILIDAD CLASE

b) Compruebe que la **probabilidad de clase** $p/(p+n)$ minimiza la suma de los errores cuadráticos.

18.9 Suponga que un algoritmo de aprendizaje intenta encontrar una hipótesis consistente cuando la clasificación de los ejemplos es en realidad aleatoria. Existen n atributos booleanos, y los ejemplos están escogidos uniformemente a partir de un conjunto de 2^n ejemplos posibles. Calcule el número de ejemplos que se requieren con anterioridad para que la probabilidad de encontrar una contradicción en los datos alcance 0,5.

18.10 Suponga que un atributo divide el conjunto de ejemplos E en subconjuntos E_i, y que cada subconjunto tiene p_i ejemplos positivos y n_i ejemplos negativos. Compruebe que el atributo tiene ganancia de información estrictamente positiva, excepto si el ratio $p_i/(p_i + n_i)$ es el mismo para todo i.

18.11 Modifique el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN para incluir la poda de X^2. Quizá quiera consultar Quinlan (1986) para ver detalles.

18.12 El algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN descrito en este capítulo no trata el caso de que algunos ejemplos carezcan de valor para ciertos atributos.

a) Primero, necesitamos encontrar una forma para poder clasificar dichos ejemplos, dado un árbol de decisión que incluya test en los atributos para los que no se tenga valor. Suponga que un ejemplo X no tiene el valor para el atributo A y que el árbol de decisión comprueba el valor de A en un nodo que X alcanza. Una forma de tratar este caso es simular que el ejemplo toma *todos* los valores posibles para el atributo, pero dando peso a cada valor de acuerdo con su frecuencia respecto a todos los ejemplos que alcanzan ese nodo en el árbol de decisión. El algoritmo de clasificación debería seguir todas las ramas desde un nodo para el cual el valor es desconocido y debería multiplicar los pesos sobre cada camino. Escriba un algoritmo de clasificación modificado para árboles de decisión que tenga dicho comportamiento.

b) Ahora modifique el cálculo de la ganancia de información para que dada cualquier colección de ejemplos C en un cierto nodo del árbol en el proceso de construcción, a los ejemplos con valores desconocidos para alguno de los atributos restantes se les dé valores de acuerdo con la frecuencia de dichos valores en el conjunto C.

18.13 En este capítulo, mostramos que los atributos con distintos valores posibles pueden provocar problemas con la medida de la ganancia. Dichos atributos tienden a dividir los ejemplos en numerosas clases pequeñas o incluso clases con un solo elemento, por ello esto parece ser extremadamente relevante de acuerdo con la medida de la ganancia. El criterio del **ratio de ganancia** selecciona atributos de acuerdo con el ratio entre su ganancia y su contenido de información intrínseco, es decir, la cantidad de información que proporciona la respuesta a la pregunta «¿Cuál es el valor de este atributo?». Por ello, el criterio del ratio de ganancia intenta medir en qué medida de eficiencia un atributo proporciona información sobre la clasificación correcta del ejemplo. Escriba una expresión matemática para la información contenida en un atributo, e implemente el criterio del ratio de ganancia para el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN.

18.14 Considere un algoritmo de aprendizaje de conjuntos de hipótesis que use el voto de la mayoría entre M hipótesis aprendidas. Suponga que cada hipótesis tiene un error ϵ y que el error producido por cada hipótesis es independiente de las otras. Calcule una fórmula para el error del algoritmo para conjuntos en términos de M y ϵ, y evalúelo para los casos en que $M = 5$, 10, y 20 y $\epsilon = 0{,}1$, 0,2 y 0,4. Si la suposición de independencia desaparece, ¿es posible que el error para conjuntos sea *peor* que ϵ?

18.15 Este ejercicio tiene que ver con la expresividad de las listas de decisión (Apartado 18.5).

a) Compruebe que las listas de decisión pueden representar cualquier función booleana, si el tamaño de los test no está limitado.

b) Compruebe que si los test pueden contener como mucho k literales cada uno, las listas de decisión pueden representar cualquier función que pueda ser representada con un árbol de decisión de profundidad k.

19 Conocimiento en el aprendizaje

En este capítulo se describe el problema del aprendizaje cuando ya se sabe algo.

En todos los enfoques sobre aprendizaje descritos en los tres capítulos anteriores la idea es construir una función que tenga el comportamiento observado en los datos, en la entrada y la salida. En cada caso, los métodos de aprendizaje se pueden entender como la búsqueda de un espacio de hipótesis para encontrar una función adecuada, partiendo únicamente de una asunción muy básica con respecto a la función, como un «polinomio de segundo grado» o un «árbol de decisión» y restricciones de la forma «cuanto más sencillo mejor». Hacer esto equivale a decir que para poder aprender algo nuevo se debe olvidar (casi) todo lo que sepa. En este capítulo, se estudian métodos de aprendizaje que pueden beneficiarse del **conocimiento *a priori*** sobre el mundo. En la mayoría de los casos, el conocimiento *a priori* se representa como teorías generales de lógica de primer orden; así, para empezar relacionaremos el trabajo de representación del conocimiento con el aprendizaje.

CONOCIMIENTO *A PRIORI*

19.1 Una formulación lógica del aprendizaje

En el Capítulo 18 se definía el aprendizaje inductivo puro como el proceso de encontrar una hipótesis que verificara los ejemplos observados. A partir de ahora, la definición se especializa al caso en el que la hipótesis se representa a través de un conjunto de sentencias lógicas. Las descripciones de los ejemplos y las clasificaciones también serán sentencias lógicas, y un nuevo ejemplo podrá ser clasificado infiriendo una sentencia de clasificación a partir de las hipótesis y de las descripciones de los ejemplos. Este enfoque tiene en cuenta la construcción incremental de la hipótesis, una sentencia cada vez.

También tiene en cuenta el conocimiento *a priori*; ya que las sentencias que ya se conocen pueden ayudar en la clasificación de nuevos ejemplos. En un principio, se puede pensar que la formulación lógica del aprendizaje supone una gran cantidad de trabajo extra, pero ayuda a aclarar muchos de los aspectos del aprendizaje. Nos permite ir más allá de los métodos de aprendizaje simples vistos en el Capítulo 18 poniendo toda la potencia de la inferencia lógica al servicio del aprendizaje.

Ejemplos e hipótesis

Recordemos el problema de aprendizaje del restaurante del Capítulo 18: aprender una regla para decidir si esperar por una mesa. Los ejemplos fueron descritos a través de **atributos** tales como *Alternativa*, *Bar*, *Vier/Sab*, etc. En un marco lógico, un ejemplo es un objeto que se describe a través de sentencias lógicas; los atributos se convierten en predicados unarios. Generalizaremos llamando al ejemplo i-ésimo X_i. Por ejemplo, el primer ejemplo de la Figura 18.3 se describe a través de la sentencia

$$Alternativa(X_1) \wedge \neg Bar(X_1) \wedge \neg Vier/Sab(X_1) \wedge Hambriento(X_1) \wedge ...$$

Usaremos la notación $D_i(X_i)$ para referirnos a la descripción de X_i, donde D_i puede ser cualquier expresión lógica con un único argumento. La clasificación del objeto viene dada por la sentencia

$$Esperar(X_1)$$

Usaremos la notación genérica $Q(X_i)$ si el ejemplo es positivo, y $\neg Q(X_i)$ si el ejemplo es negativo. El conjunto completo de entrenamiento es la conjunción de todas las sentencias de descripción y clasificación.

El propósito del aprendizaje inductivo en el marco lógico es encontrar una expresión lógica equivalente al predicado meta Q que se pueda usar para clasificar correctamente los ejemplos. Cada hipótesis propone una expresión, que denominaremos una **definición candidata** del predicado meta. Usando C_i para representar la definición candidata, cada hipótesis H_i es una sentencia de la forma $\forall x\, Q(x) \Leftrightarrow C_i(x)$. Por ejemplo, un árbol de decisión afirma que un predicado meta es verdadero para un objeto si y sólo si se alcanza una de las ramas que llevan a *verdadero*. Así, la Figura 18.6 expresa la siguiente definición lógica (a la cual denominaremos H_r para futuras referencias):

DEFINICIÓN CANDIDATA

$$\begin{aligned}\forall r\, Esperar(r) \Leftrightarrow\ & Clientes(r, Algunos)\\ \vee\ & Clientes(r, Lleno) \wedge Hambriento(r) \wedge Tipo(r, Francés)\\ \vee\ & Clientes(r, Lleno) \wedge Hambriento(r) \wedge Tipo(r, Tailandés)\\ & \quad \wedge Vier/Sab(r)\\ \vee\ & Clientes(r, Lleno) \wedge Hambriento(r) \wedge Tipo(r, Hamburgue.)\end{aligned} \tag{19.1}$$

Cada hipótesis predice que un cierto conjunto de ejemplos (por ejemplo, aquellos que satisfagan su definición candidata) serán ejemplos del predicado meta. Este conjunto se denomina **extensión** del predicado. Dos hipótesis con distintas extensiones son, por lo tanto, lógicamente inconsistentes entre sí, ya que se contradicen en sus predicciones para al menos un ejemplo. Si tienen la misma extensión, son lógicamente equivalentes.

EXTENSIÓN

El espacio de hipótesis **H** es el conjunto de todas las hipótesis $\{H_1, ..., H_n\}$ que el algoritmo de aprendizaje está diseñado para considerar. Por ejemplo, el algoritmo APREN-

DIZAJE-ÁRBOL-DECISIÓN puede considerar cualquier hipótesis del árbol de decisión definida en términos de los atributos que se proporcionan; por ello, su espacio de hipótesis está formado por todos estos árboles de decisión. Se supone que el algoritmo de aprendizaje considera que una de las hipótesis es correcta; es decir, considera la siguiente sentencia

$$H_1 \vee H_2 \vee H_3 \vee ... \vee H_n \tag{19.2}$$

Según van llegando los ejemplos, las hipótesis que no son **consistentes** con los mismos pueden ser excluidas. Examinemos esta noción de consistencia más cuidadosamente. Obviamente, si la hipótesis H_i es consistente con todo el conjunto de entrenamiento, tiene que ser consistente con cada ejemplo. ¿Qué significaría ser inconsistente con un ejemplo? Esto puede ocurrir de una de las dos siguientes formas:

FALSO NEGATIVO

- Un ejemplo puede ser un **falso negativo** para la hipótesis, si la hipótesis afirma que debe ser negativo pero en realidad es positivo. Por ejemplo, el nuevo ejemplo X_{13} descrito por:

$$Clientes(X_{13}, Lleno) \wedge TiempoEsperaEstimado(X_{13}, 0\text{-}10) \wedge \\ \wedge \neg Hambriento(X_{13}) \wedge ... \wedge Esperar(X_{13})$$

  sería un falso negativo para la hipótesis H_r dada anteriormente. A partir de H_r y la descripción del ejemplo, podemos deducir tanto $Esperar(X_{13})$, que es lo que el ejemplo afirma, como $\neg Esperar(X_{13})$, que es lo que la hipótesis predice. Por ello, la hipótesis y el ejemplo son lógicamente inconsistentes.
- Un ejemplo puede ser un **falso positivo** para la hipótesis, si la hipótesis afirma que debe ser positivo pero en realidad es negativo[1].

FALSO POSITIVO

Si un ejemplo es un falso positivo o un falso negativo para una hipótesis, el ejemplo y la hipótesis son lógicamente inconsistentes entre sí. Si asumimos que el ejemplo es una observación correcta del hecho, la hipótesis puede ser excluida. Desde un punto de vista lógico, esto es exactamente análogo a la regla de inferencia de resolución (*véase* el Capítulo 9), donde la disyunción de hipótesis corresponde a una cláusula y el ejemplo corresponde a un literal que resuelve con uno de los literales de la cláusula. Por ello, un sistema normal de inferencia lógica podría, en principio, aprender a partir del ejemplo eliminando una o más hipótesis. Suponga, por ejemplo, que la sentencia I_1 representa el ejemplo, y el espacio de hipótesis es $H_1 \vee H_2 \vee H_3 \vee H_4$. Entonces, si I_1 es inconsistente con H_2 y H_3, el sistema de inferencia lógica puede deducir el nuevo espacio de hipótesis $H_1 \vee H_4$.

Por lo tanto, podemos caracterizar el aprendizaje inductivo en un marco lógico como un proceso de eliminación gradual de las hipótesis que son inconsistentes con los ejemplos, reduciendo así las posibilidades. Ya que el espacio de hipótesis normalmente es enorme (o incluso infinito en el caso de la lógica de primer orden), no recomendamos construir un sistema de aprendizaje que demuestre teoremas usando resolución y haga una enumeración completa del espacio de hipótesis. En vez de ello, describiremos dos enfoques que encuentran hipótesis lógicamente consistentes con mucho menos esfuerzo.

[1] Los términos «falso positivo» y «falso negativo» se usan en medicina para describir resultados erróneos a partir de test de laboratorio. Un resultado es un falso positivo si indica que el paciente tiene una enfermedad cuando en realidad la enfermedad no está presente.

Búsqueda mejor-hipótesis-actual

MEJOR-HIPÓTESIS-ACTUAL

La idea de la búsqueda **mejor-hipótesis-actual** es mantener una única hipótesis, y ajustarla cuando llegan nuevos ejemplos para que se mantenga la consistencia. El algoritmo básico fue descrito por John Stuart Mill (1843), pero podría haber aparecido incluso antes.

Suponga que tenemos una hipótesis tal como H_r, con la que nos hemos encariñado bastante. Mientras que cada nuevo ejemplo sea consistente, no tenemos que hacer nada. Cuando llega un ejemplo falso negativo, X_{13}. ¿Qué debemos hacer? La Figura 19.1(a) muestra H_r esquemáticamente como una región: todo lo de dentro del rectángulo es parte de la extensión de H_r. Los ejemplos que han sido observados hasta ahora se muestran como «+» o «−», y observamos que H_r clasifica de forma correcta todos los ejemplos de *Esperar*, como ejemplos positivos o negativos. En la Figura 19.1(b), un nuevo ejemplo (con un círculo) es un falso negativo: la hipótesis dice que debería ser negativo, pero en realidad es positivo. La extensión de la hipótesis debe aumentarse para incluirlo. Esto se denomina **generalización**: una posible generalización se muestra en la Figura 19.1(c). En la Figura 19.1(d), se observa un falso positivo: la hipótesis afirma que el nuevo ejemplo (con un círculo) debería ser positivo, pero en realidad es negativo. La extensión de la hipótesis debe ser reducida para excluir al ejemplo. Esto se denomina **especialización**; en la Figura 19.1(e) se observa una posible especialización de la hipótesis.

GENERALIZACIÓN

ESPECIALIZACIÓN

Las relaciones «más general que» y «más especifica que» entre hipótesis proporcionan la estructura lógica en el espacio de hipótesis que hace posible la búsqueda eficiente.

Ahora podemos especificar el algoritmo APRENDIZAJE-MEJOR-ACTUAL, mostrado en la Figura 19.2. Nótese que cada vez que se considera una generalización o una especialización de la hipótesis, se debe comprobar la consistencia con el resto de ejemplos, ya que aumentos/reducciones de forma arbitraria en la extensión podrían incluir/excluir ejemplos negativos/positivos observados anteriormente.

Se han definido la generalización y la especialización como operaciones que modifican la *extensión* de una hipótesis. Ahora necesitamos determinar exactamente cómo se pueden implementar como operaciones sintácticas que modifiquen la definición candidata asociada con la hipótesis. Esto se realiza observando que la generalización y la es-

Figura 19.1 (a) Una hipótesis consistente. (b) Un falso negativo. (c) La hipótesis se generaliza. (d) Un falso positivo. (e) La hipótesis se especializa.

```
función APRENDIZAJE-MEJOR-ACTUAL(ejemplos) devuelve una hipótesis
  H ← cualquier hipótesis consistente con el primer ejemplo de ejemplos
  para cada ejemplo restante de ejemplos hacer
     si e es un falso positivo para H entonces
        H ← elegir una especialización de H consistente con los ejemplos
     si no si e es un falso negativo para H entonces
        H ← elegir una generalización de H consistente con los ejemplos
     si no puede encontrarse ninguna especialización/generalización consistente entonces fallo
  devolver H
```

Figura 19.2 El algoritmo de aprendizaje de la mejor hipótesis actual. Busca una hipótesis consistente y retrocede cuando no puede encontrarse ninguna especialización/generalización consistente.

pecialización también son relaciones *lógicas* entre hipótesis. Si la hipótesis H_1, con la definición C_1, es una generalización de la hipótesis H_2 con la definición C_2, entonces debemos tener

$$\forall x\ C_2(x) \Rightarrow C_1(x)$$

Por lo tanto, para construir una generalización de H_2, simplemente necesitamos encontrar la definición C_1 que sea una implicación lógica de C_2. Esto se puede hacer fácilmente. Por ejemplo, si $C_2(x)$ es $Alternativa(x) \wedge Clientes(x, Algunos)$, una posible generalización viene dada por $C_1(x) \equiv Clientes(x, Algunos)$. Esto se denomina **omisión de condiciones** (*dropping conditions*). De forma intuitiva, genera una definición más débil y por ello permite un conjunto más grande de ejemplos positivos. Existen más operaciones de generalización, dependiendo del lenguaje en el que se opere. De la misma forma, podemos especializar una hipótesis añadiendo condiciones extra a su definición candidata o eliminando disyunciones de una definición disyuntiva. Veamos cómo funciona para el ejemplo del restaurante, usando los datos de la Figura 18.3.

OMISIÓN DE CONDICIONES

- El primer ejemplo X_1 es positivo. $Alternativa(X_1)$ es verdadero, así que la hipótesis inicial será

$$H_1\text{:}\ \ \forall x\ Esperar(x) \Leftrightarrow Alternativa(x)$$

- El segundo ejemplo X_2 es negativo. H_1 lo predice como positivo, así que es un falso positivo. Por ello, tenemos que especializar H_1. Se puede hacer añadiendo una condición extra que rechace X_2. Una posibilidad es

$$H_2\text{:}\ \ \forall x\ Esperar(x) \Leftrightarrow Alternativa(x) \wedge Clientes(x, Algunos)$$

- El tercer ejemplo X_3 es positivo. H_2 lo predice como negativo, luego es un falso negativo. Por ello, necesitamos generalizar H_2. Eliminamos la condición *Alternativa*, resultando

$$H_3\text{:}\ \ \forall x\ Esperar(x) \Leftrightarrow Clientes(x, Algunos)$$

- El cuarto ejemplo X_4 es positivo. H_3 lo predice como negativo, luego es un falso negativo. Por ello, necesitamos generalizar H_3. No podemos eliminar la condición

de *Clientes*, ya que obtendríamos una hipótesis que incluye a todos los ejemplos, que sería inconsistente con X_2. Una posibilidad es añadir una disyunción:

$$H_4\text{: } \forall x\ Esperar(x) \Leftrightarrow Clientes(x, Algunos) \vee (Clientes(x, Lleno) \wedge Vier/Sab(x))$$

La hipótesis ya está comenzando a parecer razonable. Obviamente, existen otras posibilidades consistentes con los cuatro primeros ejemplos; aquí están otras dos:

$$H'_4\text{: } \forall x\ Esperar(\mathrm{x}) \Leftrightarrow \neg TiempoEsperaEstimado(x, \text{30-60})$$

$$H''_4\text{: } \forall x\ Esperar(x) \Leftrightarrow Clientes(x, Algunos) \vee$$
$$\vee (Clientes(x, Lleno) \wedge TiempoEsperaEstimado(x, \text{10-30}))$$

El algoritmo APRENDIZAJE-MEJOR-ACTUAL se describe de forma no determinística, ya que en cualquier punto, se pueden aplicar varias especializaciones o generalizaciones posibles. Las elecciones realizadas no nos llevarán necesariamente a la hipótesis más sencilla, y pueden llevarnos a una situación irrecuperable donde ninguna modificación sencilla de la hipótesis sea consistente con todos los datos. En estos casos, el programa debe volver hacia atrás (*backtracking*) al punto de elección anterior.

El algoritmo APRENDIZAJE-MEJOR-ACTUAL y sus variantes se han usado en muchos sistemas de aprendizaje automático, comenzando con el programa «*arch-learning*» de Patrick Winston (1970). Pueden aparecer algunas dificultades, con un gran número de instancias y un espacio grande:

1. La comprobación de todas las instancias anteriores cada vez que se realiza una modificación es muy cara computacionalmente.
2. El proceso de búsqueda puede conllevar un gran número de *backtrackings*. Como observamos en el Capítulo 18, el espacio de hipótesis puede ser doblemente exponencial.

Búsqueda de mínimo compromiso

El *backtracking* aparece porque el enfoque de mejor-hipótesis-actual tiene que *elegir* una hipótesis particular como su mejor suposición, aunque no tiene los suficientes datos para hacer la elección con seguridad. En vez de esto podemos mantener sólo aquellas hipótesis que son consistentes con todos los datos hasta el momento. Cada nueva instancia, o no tendrá ningún efecto o eliminará algunas de las hipótesis. Recuerde que el espacio de hipótesis original se puede ver como una disyunción de sentencias

$$H_1 \vee H_2 \vee H_3 \vee \ldots \vee H_n$$

La disyunción se reduce cuando se encuentran varias hipótesis que son inconsistentes con los datos, manteniendo sólo aquellas que no se desechan. Asumiendo que el espacio de hipótesis original contiene la respuesta correcta, la disyunción reducida seguirá conteniendo todavía la respuesta correcta, ya que sólo se han eliminado hipótesis inconsistentes. El conjunto de hipótesis restantes se denomina **espacio de versiones**, y el algoritmo de aprendizaje (mostrado esquemáticamente en la Figura 19.3) se denomina algoritmo de aprendizaje del espacio de versiones (también se conoce como el algoritmo de **eliminación del candidatos**).

ESPACIO DE VERSIONES

ELIMINACIÓN DE CANDIDATOS

```
función APRENDIZAJE-ESPACIO-VERSIONES(ejemplos) devuelve espacio de versiones
  variables locales: V, el espacio de versiones: el conjunto de todas las hipótesis

  V ← el conjunto de todas las hipótesis
  para cada ejemplo e de ejemplos hacer
      si V no está vacía entonces V ← ACTUALIZAR-ESPACIO-VERSIONES(V, e)
  devolver V
```

```
función ACTUALIZAR-ESPACIO-VERSIONES(V, e) devuelve espacio de versiones actualizado
  V ← {h ∈ V: h es consistente con e}
```

Figura 19.3 El algoritmo del espacio de versiones. Encuentra un subconjunto de V que es consistente con todos los *ejemplos*.

Una importante propiedad de este enfoque es que es *incremental*: nunca se tiene que volver hacia atrás para examinar de nuevo los antiguos ejemplos. Está garantizada la consistencia de todas las hipótesis restantes. También es un algoritmo de **mínimo-compromiso,** ya que no realiza elecciones arbitrarias (comparado con el algoritmo de planificación de orden-parcial del Capítulo 11). Pero existe un problema obvio. Ya dijimos que el espacio de hipótesis es enorme, así que ¿cómo podemos escribir esta enorme disyunción?

La siguiente analogía es muy útil. ¿Cómo se representan todos los números reales entre 1 y 2? Después de todo, ¡existe un número infinito! La solución es usar una representación de intervalos que simplemente especifique las fronteras del conjunto: [1,2]. Esto funciona porque existe un *orden* en el conjunto de los números reales.

También tenemos un orden en el espacio de hipótesis, concretamente, el que viene dado por la generalización y la especialización. Este es un orden parcial, lo que significa que cada frontera no será un punto, sino un conjunto de hipótesis que denominaremos **conjunto frontera**. El aspecto importante es que podemos representar el espacio de versiones total usando sólo dos conjuntos frontera: una frontera más general (el **conjunto *G***) y una frontera más específica (el **conjunto *S***). *Está garantizado que todo lo que se encuentre entre estos dos conjuntos será consistente con los ejemplos*. Antes de probar esto, recapitulemos:

CONJUNTO FRONTERA

CONJUNTO G

CONJUNTO S

- El espacio de versiones actual es el conjunto de las hipótesis consistentes con todos los ejemplos hasta el momento. Se representa mediante los conjuntos S y G, cada uno de los cuales es un conjunto de hipótesis.
- Cada miembro del conjunto S es consistente con todas las observaciones hasta el momento, y no existe ninguna hipótesis consistente que sea más específica.
- Cada miembro del conjunto G es consistente con todas las observaciones hasta el momento, y no existe ninguna hipótesis consistente que sea más general.

Queremos que el espacio de versiones inicial (antes de que ningún ejemplo haya sido observado) represente todas las hipótesis posibles. Conseguimos esto haciendo que el conjunto G contenga *Verdadero* (la hipótesis que contiene todo), y el conjunto S contenga *Falso* (la hipótesis cuya extensión es vacía).

La Figura 19.4 muestra la estructura general de la representación del conjunto frontera del espacio de versiones. Para mostrar que la representación es suficiente, necesitamos las dos propiedades siguientes:

1. Cada hipótesis consistente (que no esté en los conjuntos frontera) es más específica que algunos de los miembros del conjunto *G*, y más general que algunos de los miembros del conjunto *S*. (Es decir, no existen hipótesis «rezagadas» fuera.) Esto viene de la definición de *S* y *G*. Si existiera una hipótesis *h* rezagada, tendría que ser más específica que cualquiera de los miembros de *G*, en cuyo caso pertenecería a *G*; o más general que cualquiera de los miembros de *S*, en cuyo caso pertenecería a *S*.
2. Cualquier hipótesis más específica que algún miembro del conjunto *G* y más general que algún miembro del conjunto *S* es una hipótesis consistente. (Es decir, no existen «agujeros» entre las fronteras.) Cualquier *h* entre *S* y *G* debe rechazar todos los ejemplos negativos rechazados por cada miembro de *G* (ya que es más específica), y debe aceptar todos los ejemplos positivos aceptados por cualquier miembro de *S* (ya que es más general). Por ello, *h* debe verificar todos los ejemplos, y no puede ser inconsistente. La Figura 19.5 muestra la situación en la que no hay ejemplos conocidos fuera de *S*, pero sí dentro de *G*, así que cualquier hipótesis en el hueco debe ser consistente.

Hemos mostrado que *si S* y *G* se mantienen de acuerdo con sus definiciones, proporcionan una representación satisfactoria del espacio de versiones. El único problema restante es cómo *actualizar S* y *G* con un nuevo ejemplo (la labor de la función ACTUALIZAR-ESPA-

Figura 19.4 El espacio de versiones contiene todas las hipótesis consistentes con los ejemplos.

Figura 19.5 Las extensiones de los miembros de *G* y *S*. Ningún ejemplo conocido cae entre los dos conjuntos de fronteras.

CIO-VERSIONES). Esto puede que parezca algo complicado al principio, pero a partir de las definiciones y con la ayuda de la Figura 19.4, no es demasiado difícil reconstruir el algoritmo.

Necesitamos analizar los miembros S_i y G_i de los conjuntos *S* y *G*. Para cada uno, la nueva instancia puede ser un falso positivo o un falso negativo.

1. Falso positivo para S_i: esto significa que S_i es demasiado general, pero no existen especializaciones consistentes de S_i (por definición), así que la sacamos fuera del conjunto *S*.
2. Falso negativo para S_i: esto significa que S_i es demasiado específica, así que la reemplazamos por todas sus generalizaciones inmediatas, siempre que sean más específicas que algunos miembros de *G*.
3. Falso positivo para G_i: esto significa que G_i es demasiado general, así que la reemplazamos por todas sus especializaciones inmediatas, siempre que sean más generales que algunos miembros de *S*.
4. Falso negativo para G_i: esto significa que G_i es demasiado específica, pero no existen generalizaciones consistentes de G_i (por definición), así que la sacamos fuera del conjunto *G*.

Continuamos estas operaciones para cada nueva instancia hasta que ocurra:

1. Que tengamos exactamente un concepto en el espacio de versiones, en cuyo caso lo devolvemos como la única hipótesis.
2. Que el espacio de versiones se *colapse* (ni *S* ni *G* se vacían, lo que indica que no existen hipótesis consistentes con el conjunto de entrenamiento). Este es el mismo fallo que aparecía en la versión sencilla del algoritmo de los árboles de decisión.
3. Que nos quedemos sin ejemplos, con varias hipótesis restantes en el espacio de versiones. Esto significa que el espacio de versiones representa una disyunción de hipótesis. Para cada nuevo ejemplo, si todas las disyunciones están de acuer-

do, podemos devolver su clasificación. Si están en desacuerdo, una posibilidad es tomar el voto de la mayoría.

Dejamos como ejercicio la aplicación del algoritmo de APRENDIZAJE-ESPACIO-VERSIONES a los datos del restaurante.

Existen dos desventajas principales del enfoque del espacio de versiones:

- Si el dominio contiene ruido o atributos insuficientes para la clasificación exacta, el espacio de versiones siempre se colapsará.
- Si permitimos disyunciones ilimitadas en el espacio de hipótesis, el conjunto S siempre contendrá una única hipótesis más específica, a saber, la disyunción de las descripciones de los ejemplos positivos hasta el momento. Del mismo modo, el conjunto G contendrá sólo la negación de la disyunción de las descripciones de los ejemplos negativos.
- Para algunos espacios de hipótesis, el número de elementos en el conjunto G y S pueden crecer exponencialmente con respecto al número de atributos, incluso aunque existan algoritmos eficientes de aprendizaje para dichos espacios de hipótesis.

Hasta la fecha, no se ha encontrado ninguna solución completa y satisfactoria para el problema del ruido. El problema de las disyunciones puede ser abordado permitiendo formas limitadas de disyunciones o incluyendo una **jerarquía de generalización** de predicados más generales. Por ejemplo, en vez de usar la disyunción *TiempoEsperaEstimado*(x, 30-60) $\vee$ *TiempoEsperaEstimado*(x, >60), deberíamos usar el único literal *LargaEspera*(x). El conjunto de operaciones de generalización y especialización puede ser fácilmente ampliado para dar soporte a esto.

JERARQUÍA DE GENERALIZACIÓN

El algoritmo de espacio de versiones puro fue aplicado inicialmente en el sistema META-DENDRAL, que fue diseñado para aprender reglas para predecir cómo las moléculas se rompían en trozos en un espectómetro de masas (Buchanan y Mitchell, 1978). META-DENDRAL fue capaz de generar reglas que fueron lo suficientemente novedosas como para garantizar publicaciones en una revista de química analítica (el primer conocimiento realmente científico generado por un programa informático). También se aplicó en el elegante sistema LEX (Mitchell *et al.*, 1983), que era capaz de aprender a resolver problemas de integración simbólica estudiando sus propios éxitos y fracasos. Aunque, probablemente, los métodos del espacio de versiones no son prácticos en la mayoría de problemas de aprendizaje, en gran medida por el ruido, proporcionan un buen tratamiento de los espacios de hipótesis con estructura lógica.

19.2 Conocimiento en el aprendizaje

La sección anterior describía el marco más simple para la lógica inductiva. Para entender el papel del conocimiento *a priori*, necesitamos hablar de las relaciones lógicas entre hipótesis, descripciones de ejemplos y clasificaciones. Mediante *Descripciones* denotaremos la conjunción de todas las descripciones de los ejemplos del

conjunto de entrenamiento, y mediante *Clasificaciones* la conjunción de todas las clasificaciones de los ejemplos. De este modo, las *Hipótesis* que «explican las observaciones» deben satisfacer la siguiente propiedad (recuerde que $\models$ significa «es consecuencia lógica»):

$$Hipótesis \wedge Descripciones \models Clasificaciones \tag{19.3}$$

RESTRICCIÓN

A este tipo de relación lo denominaremos **restricción**, donde *Hipótesis* es lo «desconocido». El objetivo del aprendizaje inductivo puro es resolver esta restricción, donde *Hipótesis* se obtiene a partir de algún espacio de hipótesis predefinido. Por ejemplo, si consideramos un árbol de decisión como una fórmula lógica (*véase* Ecuación (19.1) en la página 774), un árbol de decisión consistente con todos los ejemplos satisfará la Ecuación (19.3). Por supuesto, si no se ponen restricciones sobre la forma lógica que adopta la hipótesis, *Hipótesis* = *Clasificaciones* también satisface la restricción. La navaja de Ockham indica que de las hipótesis consistentes son preferibles las más *pequeñas*, por lo tanto intentaremos hacer algo mejor que simplemente memorizar ejemplos.

Este esquema simple del aprendizaje inductivo persiste desde los inicios de 1980. El enfoque moderno consiste en diseñar agentes que *ya saben algo* e intentan aprender más. Esto no supone una gran perspicacia, pero hace bastante diferente la forma de diseñar agentes. También puede tener alguna relevancia en nuestras teorías sobre cómo funciona la ciencia. La idea general se muestra esquemáticamente en la Figura 19.6.

Si queremos construir un agente de aprendizaje autónomo que utilice conocimiento básico, en primer lugar el agente debe contar con algún método para obtener el conocimiento básico, para que este conocimiento se pueda utilizar en los nuevos episodios de aprendizaje. Este método también debe ser en sí mismo un proceso de aprendizaje. Por lo tanto, la historia de la vida del agente estará caracterizada por un desarrollo *acumulativo* o *incremental*. Presumiblemente, el agente podría empezar sin nada, realizando inducciones *in vacuo*, como un pequeño programa de inducción pura. Pero una vez haya comido del Árbol del Conocimiento no podrá continuar con simples especulaciones y deberá utilizar su conocimiento básico para aprender más y de una forma más efectiva. La pregunta es cuándo hacer esto realmente.

Figura 19.6 Un proceso de aprendizaje acumulativo utiliza y amplía un almacén de conocimiento básico a medida que pasa el tiempo.

Algunos ejemplos sencillos

Consideremos algunos ejemplos de aprendizaje con conocimiento básico de sentido común. Muchas inferencias, aparentemente racionales, que se realizan a la vista de observaciones, claramente no siguen los principios de la inducción pura:

- Algunas veces se sacan conclusiones generales después de una única observación. Gary Larson una vez pintó un dibujo animado en el que un hombre primitivo con gafas, Zog, está asando un lagarto en una vara. Es observado por una multitud asombrada de sus contemporáneos menos intelectuales, quienes han venido utilizando sus manos desnudas para mantener sus provisiones sobre el fuego. Esta instructiva experiencia es suficiente para convencer a los observadores de un principio general para cocinar sin dolor.
- Considere el caso de un turista que va a Brasil y conoce a su primera persona brasileña. Al escucharle hablar en portugués, inmediatamente concluirá que los brasileños hablan en portugués, aunque al descubrir que su nombre es Fernando no concluirá que todos los brasileños se llaman Fernando. En la ciencia existen casos similares. Por ejemplo, cuando un estudiante de primer año de Física mide la densidad y la conductancia de una muestra de cobre a una temperatura determinada, está bastante seguro cuando generaliza estos valores a todas las piezas de cobre. Aunque cuando mida la masa, no considerará la hipótesis de que todas las piezas de cobre tienen esa masa. Por otro lado, sería bastante razonable hacer este tipo de generalización sobre todos los peniques.
- Finalmente, considere el caso de un estudiante de Medicina que sabe realizar diagnósticos sofisticados, pero es ignorante respecto a farmacología, que está observando la consulta entre un paciente y un experto. Después de una serie de preguntas y respuestas, el experto dice al paciente que tome un determinado antibiótico. El estudiante de medicina inferirá la regla general sobre qué antibiótico particular es efectivo para un tipo de infección determinado.

En todos estos casos *el uso de conocimiento básico permite un aprendizaje mucho más rápido que el que se podría esperar de un programa de inducción puro*.

Algunos esquemas generales

En cada uno de los ejemplos anteriores, se puede apelar al conocimiento *a priori* para intentar justificar las generalizaciones realizadas. Veremos qué tipos de restricciones se generan en cada caso. Las restricciones incluirán el *ConocimientoDeBase*, además de las *Hipótesis*, las *Descripciones* observadas y las *Clasificaciones*.

En el caso del lagarto asado, el hombre primitivo generaliza *explicando* el éxito de la vara: ésta soporta al lagarto, manteniendo así la mano lejos del fuego. A partir de esta explicación, infiere una regla general: que cualquier objeto largo, rígido y puntiagudo sirve para asar alimentos pequeños y ligeros. Este tipo de proceso de generalización ha sido denominado **aprendizaje basado en explicaciones**, o **EBL** *(Explanation Based Learning)*. Observe que la regla general se *deduce lógicamente* del conocimiento básico

APRENDIZAJE BASADO EN EXPLICACIONES

que posee el hombre primitivo. Por lo tanto las restricciones que se satisfacen por el EBL son las siguientes:

$$Hipótesis \wedge Descripciones \models Clasificaciones$$
$$ConocimientoBásico \models Hipótesis$$

El hecho de que EBL utilice la Ecuación (19.3), en un principio hizo suponer que era una forma mejor de aprender a partir de ejemplos. Pero como necesita que exista suficiente conocimiento básico como para explicar la *Hipótesis*, que en realidad explica las observaciones, *realmente el agente no aprende nada nuevo del ejemplo*. El agente podría haber derivado el ejemplo de lo que ya sabía, aunque esto requiriera una cantidad muy elevada de computación. EBL no se considera como un método para convertir los primeros principios de las teorías en conocimiento útil de propósito específico. Describiremos los algoritmos para EBL en el Apartado 19.3.

La situación del turista en Brasil es bastante diferente, puede que éste no se explique por qué Fernando habla de esa forma, a menos que conozca sus bulas Papales. Además, se puede esperar la misma generalización de un turista completamente ignorante en lo relativo a historia colonial. En este caso, el conocimiento *a priori* relevante es que, en cualquier país, la mayoría de la gente habla el mismo idioma; por otro lado, no se asume que Fernando sea el nombre de todos los brasileños, porque este tipo de regularidad no se verifica para los nombres. De forma similar, el estudiante de primer año de Física también debería tener problemas para explicar sus descubrimientos acerca de los valores particulares para la conductancia y la densidad del cobre. Sin embargo, sabe que el material del que un objeto está compuesto, junto con su temperatura, determinan su conductancia. En cada caso, el C*onocimientoBásico a priori* se refiere a la **relevancia** que tiene un conjunto de características sobre el predicado meta. Este conocimiento, *junto con las observaciones*, permite al agente inferir una nueva regla general que explica las observaciones:

RELEVANCIA

$$Hipótesis \wedge Descripciones \models Clasificaciones$$
$$Hipótesis \wedge Descripciones \wedge Clasificaciones \models Hipótesis \qquad (19.4)$$

APRENDIZAJE BASADO EN RELEVANCIA

Denominaremos a este tipo de generalización **aprendizaje basado en relevancia** o **RBL** (*Relevance-Based Learning*) (aunque este nombre no es estándar). Observe que como el RBL no hace uso del contenido de las observaciones, no produce hipótesis que vayan más allá del contenido lógico del conocimiento básico y de las observaciones. Es un método de aprendizaje *deductivo*, y por sí mismo no puede justificar la creación de nuevo conocimiento sin un conocimiento inicial de partida.

En el caso del estudiante de Medicina que observa al experto, asumimos que el conocimiento *a priori* del estudiante es suficiente para inferir la enfermedad *D* del paciente a partir de los síntomas. Sin embargo esto no es suficiente para explicar el hecho de que el doctor prescriba una medicina concreta *M*. El estudiante necesita obtener otra regla: que *M*, generalmente, es una medicina efectiva contra la enfermedad *D*. Dada esta regla, y el conocimiento *a priori* con el que cuenta el estudiante, éste puede explicar por qué el experto prescribe *M* en ese caso particular. Podemos generalizar este ejemplo para proponer la restricción:

$$ConocimientoBásico \wedge Hipótesis \wedge Descripciones \models Clasificaciones \qquad (19.5)$$

Es decir, *el conocimiento básico y la nueva hipótesis se combinan para explicar los ejemplos*. Al igual que el aprendizaje inductivo puro, el algoritmo de aprendizaje debe proponer hipótesis tan simples como sea posible, que sean consistentes con esta restricción. Los algoritmos que satisfacen la restricción (19.5) se denominan algoritmos de **aprendizaje inductivo basado en el conocimiento**, o algoritmos **KBIL** (*Knowledge-Based Inductive Learning*).

APRENDIZAJE INDUCTIVO BASADO EN EL CONOCIMIENTO

PROGRAMACIÓN LÓGICA INDUCTIVA

Los algoritmos KBIL, que se describen en detalle en el Apartado 19.5, se han estudiado principalmente en el campo de la **programación lógica inductiva** o **ILP**. En los sistemas ILP, el conocimiento *a priori* juega dos papeles clave en la reducción de la complejidad del aprendizaje:

1. El tamaño efectivo del espacio de hipótesis se reduce para incluir únicamente las teorías que son consistentes con lo que ya se conoce, porque cualquier hipótesis generada debe ser consistente tanto con el conocimiento *a priori* como con las nuevas observaciones.
2. Para un conjunto dado de observaciones, el tamaño de la hipótesis que se requiere para construir una explicación de las observaciones se debe reducir, porque el conocimiento *a priori* estará disponible para ayudar a las nuevas reglas en la explicación de las observaciones. Cuanto más pequeña sea la hipótesis, más fácil será encontrarla.

Adicionalmente, para permitir el uso de conocimiento *a priori* en la inducción, los sistemas ILP pueden formular hipótesis en lógica de primer orden general, en vez de utilizar el lenguaje restringido basado en atributos del Capítulo 18. Esto significa que pueden aprender en entornos que no pueden ser entendidos por sistemas más simples.

19.3 Aprendizaje basado en explicaciones

Como hemos explicado en la introducción de este capítulo, el aprendizaje basado en explicaciones es un método para extraer reglas generales a partir de observaciones particulares. Por ejemplo, considere el problema de diferenciar y simplificar expresiones algebraicas (Ejercicio 9.15). Si diferenciamos la expresión X^2, respecto a X, obtenemos $2X$. (Utilizamos una letra mayúscula para la variable desconocida, para distinguirla de la variable lógica x.) En un sistema de razonamiento lógico, el objetivo se puede expresar como PREGUNTA($Derivada(X^2, X) = d, BC$), con solución $d = 2X$.

Cualquier persona que conozca el cálculo diferencial puede ver esta solución «por inspección» como un resultado de la práctica en la resolución de este tipo de problemas. Un estudiante que se encuentra con este tipo de problemas por primera vez, o un programa sin experiencia, tendrá que realizar un trabajo mucho más difícil. La aplicación de las reglas estándar de diferenciación al final produce la expresión $1 \times (2 \times (X^{(2-1)}))$, y esto se acaba simplificando a $2X$. En la implementación con programación lógica de los autores, esto da lugar a 136 pasos de prueba, de los cuales 99 se producen sobre ramas finales en la prueba. Después de esta experiencia, nos gustaría que el programa resolviera el mismo problema mucho más rápido la siguiente vez que surja.

MEMORIZACIÓN

La técnica de **memorización** ha sido extensamente utilizada en informática para acelerar los programas guardando los resultados de la computación. La idea básica de las funciones memo es acumular una base de datos de pares entrada/salida; cuando se hace una llamada a la función, ésta, en primer lugar, comprueba la base de datos para ver si se puede evitar que la resolución del problema parta de cero. El aprendizaje basado en explicaciones lleva esta idea más allá, creando reglas *generales* que cubren una clase completa de casos. En el caso de la diferenciación, la memorización permitiría recordar que la derivada de X^2 es $2X$, pero si el agente tuviera que calcular la derivada de Z^2 respecto a Z, lo haría a partir de cero. Nos gustaría ser capaces de extraer la regla general[2] de que para cualquier variable desconocida u, la derivada de u^2 respecto a u es $2u$. En términos lógicos, esto se expresa mediante la regla:

$$ExpresionDesconocida(u) \Rightarrow Derivada(u^2, u) = 2u$$

Si la base de conocimiento contiene esta regla, cualquier instancia de ella podrá resolverse de forma inmediata.

Por supuesto, éste es simplemente un ejemplo trivial de un fenómeno muy general. Una vez que algo se entiende, puede ser generalizado y reutilizado en otras circunstancias. Se convierte en un paso «obvio» que, posteriormente, puede utilizarse como un bloque para resolver problemas todavía más complejos. Alfred North Whitehead (1911), coautor con Bertrand Russell de *Principia Mathematica*, escribió: «*La Civilización avanza extendiendo el número de operaciones importantes que puede realizar sin necesidad de pensar*», quizás él también aplicó EBL a su entendimiento de los acontecimientos, como en el descubrimiento de Zog. Si el lector ha entendido la idea básica del ejemplo de la diferenciación, su cerebro ya estará ocupado intentando extraer los principios generales del aprendizaje basado en explicaciones a partir de él. Observe que *todavía* no había descubierto el EBL antes de que viera el ejemplo. Al igual que los hombres primitivos observando a Zog, necesitábamos un ejemplo antes de que pudiéramos generar los principios básicos. El motivo de que ocurra esto es que *explicar por qué* algo es una buena idea es mucho más sencillo que plantearse la idea en primer lugar.

Extraer reglas generales a partir de ejemplos

La idea en la que se basa EBL consiste en construir, en primer lugar, una explicación de la observación, utilizando conocimiento *a priori*, y posteriormente, establecer una definición de la clase de casos para los que se puede utilizar la misma estructura de explicación. Esta definición proporciona las bases para una regla que cubra todos los casos en la clase. La «explicación» puede ser una prueba lógica, aunque de forma más general, puede ser cualquier proceso de razonamiento o de resolución de problemas cuyos pasos estén bien definidos. La clave es ser capaz de identificar las condiciones necesarias para que los mismos pasos se puedan aplicar a otro caso.

[2] Por supuesto, también se puede producir una regla general para u^n, pero el ejemplo actual es suficiente para este punto.

Para nuestro sistema de razonamiento utilizaremos el demostrador de teoremas con encadenamiento hacia atrás simple que se describió en el Capítulo 9. El árbol de prueba para $Derivada(X^2, X) = 2X$ es muy grande para utilizarlo como ejemplo, así que nos basaremos en un problema más sencillo para ilustrar el método de generalización. Suponga que nuestro problema consiste en simplificar $1 \times (0 + X)$. La base de conocimiento incluye las siguientes reglas:

$$Reescribir(u, v) \wedge Simplificar(v, w) \Rightarrow Simplificar(u, w)$$
$$Primitiva(u) \Rightarrow Simplificar(u, u)$$
$$ExpresionDesconocida(u) \Rightarrow Primitiva(u)$$
$$Numero(u) \Rightarrow Primitiva(u)$$
$$Reescribir(1 \times u, u)$$
$$Reescribir(0 + u, u)$$
$$\vdots$$

La prueba de que la respuesta es X se muestra en la parte de arriba de la Figura 19.7. Realmente, el método EBL construye simultáneamente dos árboles de prueba. El segundo árbol utiliza una meta *variabilizada,* en la que las constantes de la meta original han sido

Figura 19.7 Árboles de prueba para el problema de simplificación. El primer árbol muestra la prueba para el problema original, del que se puede derivar

$$ExpresionDesconocida(z) \Rightarrow Simplificar(1 \times (0 + z), z)$$

El segundo árbol muestra la prueba para un problema en el que todas las constantes se han sustituido por variables, de este árbol se pueden derivar otras reglas.

sustituidas por variables. Como la prueba original tiene éxito, la prueba con la meta variabilizada también lo tiene, *aplicando exactamente las mismas reglas*. Esto podría causar que algunas de las variables se instanciaran. Por ejemplo, para usar la regla *Reescribir*($1 \times u$, u), la variable x en la submeta *Reescribir*($x \times (y + z)$, v), debe tomar el valor 1. De la misma forma, y debe tomar el valor 0 en la submeta *Reescribir*($y + z$, v'), para utilizar la regla *Reescribir*($0 + u$, u). Una vez que hemos generalizado el árbol de prueba, tomamos las hojas (con las asignaciones necesarias) y formamos una regla general para el predicado meta:

$$Reescribir(1 \times (0 + z), 0 + z) \wedge Reescribir(0 + z, z) \wedge ExpresionDesconocida(z) \Rightarrow \\ \Rightarrow Simplificar(1 \times (0 + z), z)$$

Observe que las dos primeras condiciones de la parte izquierda son verdaderas *independientemente del valor* de z y, por lo tanto, las podemos eliminar:

$$ExpresionDesconocida(z) \Rightarrow Simplificar(1 \times (0 + z), z)$$

En general, se pueden eliminar condiciones de la regla final, si éstas no imponen restricciones sobre las variables de la parte derecha de la regla, porque la regla resultante seguirá siendo verdadera y además será más eficiente. Observe que no es posible eliminar la condición *ExpresionDesconocida*(z), porque no todos los posibles valores de z son desconocidos. Los valores que no son desconocidos requieren simplificaciones: por ejemplo, si z era 2×3, la simplificación correcta de $1 \times (0 + (2 \times 3))$ sería 6 y no 2×3.

Para recapitular, el proceso básico de EBL funciona de la siguiente forma:

1. Dado un ejemplo, se construye una prueba para el predicado meta que se aplica al ejemplo, utilizando el conocimiento básico disponible.
2. En paralelo, se construye un árbol de prueba generalizado para la meta variabilizada, utilizando los mismos pasos de inferencia que en la prueba original.
3. Se construye una nueva regla cuya parte izquierda está formada por las hojas del árbol de prueba y cuya parte derecha es la meta variabilizada (después de aplicar las asignaciones necesarias de la prueba generalizada).
4. Se eliminan todas las condiciones que son verdaderas independientemente de los valores de las variables de la meta.

Mejorar la eficiencia

El árbol de prueba generalizado de la Figura 19.7 realmente da lugar a más de una regla generalizada. Por ejemplo, si finalizamos, o **podamos**, el desarrollo de la parte derecha del árbol, cuando se llega al paso *Primitiva*, tendríamos la regla:

$$Primitiva(z) \Rightarrow Simplificar(1 \times (0 + z), z)$$

Esta regla es igualmente válida, pero más general que la regla con *ExpresionDesconocida*, porque cubre los casos en los que z es un número. Se pueden extraer reglas todavía más generales, podando después del paso *Simplificar*($y + z$, w) obtendríamos la regla

$$Simplificar(y + z, w) \Rightarrow Simplificar(1 \times (y + z), w)$$

En general, se puede extraer una regla de cada *subárbol parcial* del árbol de prueba generalizado. El problema ahora es: ¿cuál de estas reglas elegimos?

La decisión sobre qué regla generar viene determinada por la eficiencia. Existen tres factores implicados en el análisis de la eficiencia para EBL:

1. Añadir un gran número de reglas puede ralentizar el proceso de razonamiento, porque el mecanismo de inferencia debe comprobar las reglas incluso en los casos en los que no llevan a una solución. En otras palabras, se incrementa el **factor de ramificación** del espacio de búsqueda.
2. Para compensar el descenso de velocidad en el razonamiento, las reglas obtenidas en los casos que cubren deben dar lugar a incrementos significativos en la velocidad. Principalmente, estos incrementos se producen porque las reglas obtenidas evitan ramas que conducen a fracaso, que en otro caso podrían haberse tomado, pero también porque reducen la propia prueba.
3. Las reglas obtenidas deben ser tan genéricas como sea posible, para que se puedan aplicar al mayor número posible de casos.

OPERACIONALIDAD

Un enfoque muy común para asegurar que las reglas obtenidas son eficientes es insistir en la **operacionalidad** de cada submeta de la regla. Una submeta es operacional si es «fácil» de resolver. Por ejemplo, la submeta *Primitiva*(z) es fácil de resolver, ya que requiere como máximo dos pasos, mientras que la submeta *Simplificar*($y + z$, w) puede dar lugar a una cantidad de inferencia arbitraria, dependiendo de los valores de y y z. Si en cada paso de la construcción de la prueba generalizada llevamos a cabo un test sobre la operacionalidad, podremos podar lo que falta de una rama tan pronto como encontremos una submeta operacional, manteniendo la meta operacional como una conjunción en la nueva regla.

Desgraciadamente, existe un compromiso entre operacionalidad y generalidad. Normalmente, las submetas más específicas son más fáciles de resolver, pero cubren pocos casos. También es cierto que la operacionalidad es una cuestión de grado: tener que realizar uno o dos pasos es algo claramente operacional, pero ¿qué podemos decir respecto a la operacionalidad para 10 o 100 pasos? Finalmente, el coste de resolver una submeta dada depende de las otras reglas que contiene la base de conocimiento. Puede aumentar o disminuir a medida que se van añadiendo más reglas. Por este motivo, los sistemas de EBL se enfrentan realmente a un problema de optimización muy complejo, cuando intentan maximizar la eficiencia de una base de conocimiento inicial dada. En algunas ocasiones, es posible derivar un modelo matemático del efecto que produce la adición de una determinada regla en la eficiencia global, para posteriormente utilizar este modelo a la hora de seleccionar la mejor regla. El análisis puede resultar muy complicado, especialmente cuando hay reglas recursivas implicadas. Un enfoque prometedor, consiste en solucionar el problema de la eficiencia de una forma empírica, es decir, simplemente se añaden varias reglas y se observa cuáles de ellas son útiles y realmente aceleran el proceso.

En realidad, el análisis empírico de la eficiencia es el corazón del EBL. Lo que hemos denominado la «eficiencia de una base de conocimiento» es realmente la complejidad, para el caso medio, de una distribución de problemas, *generalizando a partir de ejemplos de problemas pasados, el EBL hace la base de conocimiento más eficiente, para*

el tipo de problemas que es razonable esperar. A base de realizar este trabajo, la distribución de ejemplos pasados se va aproximando más a la de ejemplos futuros (la misma suposición que se utilizaba en el PAC-aprendizaje en el Apartado 18.5). Si el sistema EBL se diseña cuidadosamente, es posible obtener velocidades significativas. Por ejemplo, un sistema de lenguaje natural basado en Prolog, diseñado para traducción de habla entre sueco e inglés fue capaz de conseguir un buen funcionamiento en tiempo real, gracias a la aplicación del EBL en el proceso del analizador sintáctico (Samuelsson y Rayner, 1991).

19.4 Aprendizaje basado en información relevante

Parece que nuestro turista en Brasil es capaz de hacer una generalización segura sobre el idioma que hablan otros brasileños. Su conocimiento básico autoriza la inferencia: la gente del mismo país (normalmente) habla el mismo idioma. Esto se puede expresar en lógica de primer orden de la siguiente manera[3]:

$$Nacionalidad(x, n) \wedge Nacionalidad(y, n) \wedge Idioma(x, l) \Rightarrow Idioma(y, l) \qquad (19.6)$$

(Traducción literal: «si x e y tienen la misma nacionalidad n, y x habla el idioma l, entonces y también habla este idioma.») No es difícil demostrar que a partir de esta sentencia y de la observación,

$$Nacionalidad(Fernando, Brasil) \wedge Idioma(Fernando, Portugues)$$

se deduce la siguiente conclusión (ver Ejercicio 19.1).

$$Nacionalidad(x, Brasil) \Rightarrow Idioma(x, Portugues)$$

Las sentencias como la (19.6) expresan una forma estricta de relevancia: dada una nacionalidad, el idioma queda totalmente determinado. (Dicho de otro modo: el idioma es función de la nacionalidad.) Estas sentencias se denominan **dependencias funcionales** o **determinaciones**. Son tan comunes en algunos tipos de aplicaciones, (por ejemplo, definiendo diseños de bases de datos) que se utiliza una sintaxis especial para escribirlas. Adoptaremos la notación de Davies (1985):

DEPENDENCIAS FUNCIONALES

DETERMINACIONES

$$Nacionalidad(x, n) \succ Idioma(x, l)$$

Esto es simplemente el *azúcar sintáctico* al que ya estamos acostumbrados, pero deja claro que la determinación es realmente una relación entre los predicados: la nacionalidad determina el lenguaje. Las propiedades relevantes que determinan la conductancia y la densidad se pueden expresar de una forma similar:

$$Material(x, m) \wedge Temperatura(x, t) \succ Conductancia(x, p);$$
$$Material(x, m) \wedge Temperatura(x, t) \succ Densidad(x, d)$$

[3] Para simplificar asumimos que una persona sólo habla un idioma. Obviamente, la regla también debería ser enmendada para países como Suiza o India.

Las generalizaciones correspondientes se obtienen lógicamente a partir de las determinaciones y de las observaciones.

Determinar el espacio de hipótesis

Aunque las determinaciones autorizan la obtención de conclusiones generales para todos los brasileños, o para todas las piezas de cobre a una temperatura dada, por supuesto, no pueden dar lugar a una teoría de *predicción* general para *todas* las nacionalidades, o para *todas* las temperaturas y materiales, a partir de un simple ejemplo. Se puede decir que su principal efecto es que limitan el espacio de hipótesis que el agente de aprendizaje necesita considerar. Por ejemplo, para predecir la conductancia, se necesita considerar únicamente el material y la temperatura, y se puede ignorar la masa, el propietario, el día de la semana, el presidente actual, etc. Desde luego, las hipótesis pueden incluir términos que vienen determinados por el material y la temperatura, como la estructura molecular, la energía térmica, o la densidad de electrones libres. *Las determinaciones especifican un vocabulario básico suficiente para construir hipótesis relativas al predicado principal*. Esta sentencia se puede demostrar mostrando que una determinación dada es lógicamente equivalente a una sentencia en la que el predicado principal es uno de los elementos del conjunto de todas las definiciones que se pueden expresar utilizando los predicados de la parte derecha de la determinación.

Intuitivamente, está claro que una reducción en el tamaño del espacio de hipótesis debería hacer más fácil el aprendizaje del predicado principal. Utilizando los resultados básicos de la teoría computacional del aprendizaje (Apartado 18.5), podemos cuantificar las ganancias posibles. Primero, recuerde que para las funciones Booleanas, se requieren $\log(|\mathbf{H}|)$ ejemplos, donde $|\mathbf{H}|$ es el tamaño del espacio de hipótesis, para que haya convergencia hacia una hipótesis razonable. Si el aprendiz cuenta con n características booleanas para construir las hipótesis, en ausencia de restricciones adicionales, $|\mathbf{H}| = O(2^{2^n})$, por lo que el número de ejemplos es $O(2^n)$. Si la determinación contiene d predicados en su parte izquierda, el aprendiz necesitará sólo $O(2^d)$ ejemplos, una reducción de $O(2^{n-d})$. Para espacios de hipótesis restringidos, como un espacio restringido de forma conjuntiva, la reducción será menos dramática, pero seguirá siendo significativa.

Aprender y utilizar información relevante

Como ya comentamos en la introducción de este capítulo, el conocimiento *a priori* es muy útil para el aprendizaje, pero también tiene que aprenderse. Por lo tanto, para conocer completamente el aprendizaje basado en información relevante, debemos proporcionar un algoritmo de aprendizaje para determinaciones. El algoritmo de aprendizaje que vamos a presentar se basa en un intento sencillo para encontrar la determinación más simple que es consistente con las observaciones. Una determinación $P \succ Q$ indica que si algunos ejemplos encajan con P, también deben encajar con Q. Por lo tanto, una determinación es consistente con un conjunto de ejemplos, si cada par que encaja con los predicados de la parte izquierda también encaja con el predicado principal, es decir, tie-

ne la misma clasificación. Por ejemplo, suponga que tenemos los siguientes ejemplos, relativos a medidas de conductancia en unas muestras de material:

Muestra	Masa	Temperatura	Material	Tamaño	Conductancia
S1	12	26	Cobre	3	0,59
S1	12	100	Cobre	3	0,57
S2	24	26	Cobre	6	0,59
S3	12	26	Plomo	2	0,05
S3	12	100	Plomo	2	0,04
S4	24	26	Plomo	4	0,05

La determinación consistente mínima es *Material* $\wedge$ *Temperatura* $\succ$ *Conductancia*. Existe una determinación no mínima, pero consistente: *Masa* $\wedge$ *Tamaño* $\wedge$ *Temperatura* $\succ$ *Conductancia*. Es consistente con los ejemplos porque la masa y el tamaño determinan la densidad, y en nuestro conjunto de datos no tenemos dos materiales diferentes con la misma densidad. Como es usual, necesitaríamos una muestra más grande para eliminar una hipótesis casi correcta.

Existen varios algoritmos para encontrar determinaciones consistentes mínimas. El enfoque más obvio es hacer una búsqueda en el espacio de las determinaciones, comprobando todas las determinaciones con un predicado, dos predicados, etc., hasta que se encuentre una determinación consistente. Asumiremos una representación sencilla basada en atributos, como la que se utilizó en el aprendizaje de árboles de decisión del Capítulo 18. Una determinación *d* se representará mediante el conjunto de atributos de la parte izquierda, porque se asume que el predicado principal es fijo. El algoritmo básico se puede observar en la Figura 19.8.

```
función DET-CONSISTENTE-MINIMA(E, A) devuelve un conjunto de atributos
   entrada: E, un conjunto de ejemplos
            A, un conjunto de atributos de tamaño n

   para i = 0, ..., n hacer
      para cada subconjunto A_i de A de tamaño i hacer
         si DET-CONSISTENTE?(A_i, E) entonces devolver A_i
```

```
función DET-CONSISTENTE? (A, E) devuelve un valor de verdad
   entrada: A, un conjunto de atributos
            E, un conjunto de ejemplos
   variables locales: H, una tabla hash

   para cada ejemplo e en E hacer
      si algún ejemplo de H tiene los mismos valores que e para los atributos A, pero
         diferente clasificación entonces devolver falso
      guardar la clase de e en H, indexada por los valores para los atributos A del ejemplo e
devolver verdadero
```

Figura 19.8 Un algoritmo para encontrar la determinación consistente mínima.

La complejidad en tiempo de este algoritmo depende del tamaño de la determinación más pequeña que sea consistente. Suponiendo que esta determinación tiene p atributos de n atributos totales, el algoritmo no la encontrará hasta que busque en los subconjuntos de A de tamaño p. Existen $\binom{n}{p} = O(n^p)$ subconjuntos de este tamaño; así que el algoritmo es exponencial en el tamaño de la determinación mínima. Por lo tanto, como el problema resulta ser NP-completo, no se puede esperar que en el caso general sea mejor. Sin embargo, la mayoría de los dominios tendrán la suficiente estructura local (*véase* Capítulo 14 para una definición de dominios localmente estructurados) para que p sea pequeña.

Dado un algoritmo para el aprendizaje de determinaciones, un agente de aprendizaje cuenta con una forma de construir una hipótesis mínima con la que aprender el predicado meta. Por ejemplo, podemos combinar DET-CONSISTENTE-MÍNIMA con el algoritmo APRENDIZAJE-ÁRBOL-DECISIÓN. Esto lleva a un algoritmo de aprendizaje de árboles de decisión basado en relevancia (RBDTL), que primero identifica el conjunto mínimo de atributos relevantes y después pasa este conjunto al algoritmo de aprendizaje del árbol de decisión. Al contrario que APRENDIZAJE-ÁRBOL-DECISIÓN, RBDTL simultáneamente aprende y utiliza información relevante para minimizar el espacio de hipótesis. Se espera que RBDTL aprenda más rápidamente que APRENDIZAJE-ÁRBOL-DECISIÓN, de hecho así es. La Figura 19.9 muestra el rendimiento del aprendizaje de los dos algoritmos, sobre datos generados aleatoriamente para una función que depende solo de cinco de 16 atributos. Obviamente, en los casos en que todos los atributos son relevantes, RBDTL no supone ninguna ventaja.

RESTRICCIONES DECLARATIVAS

Esta sección únicamente introduce el campo de las **restricciones declarativas**, que ayuda a entender cómo se puede utilizar el conocimiento *a priori* para identificar el espacio de hipótesis apropiado para buscar la definición meta correcta. Aún quedan muchas preguntas sin respuesta:

Figura 19.9 Comparación entre el rendimiento de RBDTL y APRENDIZAJE-ÁRBOL-DECISIÓN, realizada sobre datos generados aleatoriamente para la función meta, que depende exclusivamente de cinco de 16 atributos.

- ¿Cómo se pueden extender los algoritmos para manejar el ruido?
- ¿Cómo manejar variables con valor continuo?
- ¿Cómo se pueden utilizar otros tipos de conocimiento *a priori*, además de las determinaciones?
- ¿Cómo se pueden generalizar los algoritmos para que puedan trabajar con cualquier teoría de primer orden, en vez de únicamente con una representación basada en atributos?

Algunas de estas cuestiones se responden en la siguiente sección.

19.5 Programación lógica inductiva

La programación lógica inductiva (ILP) combina los métodos de inducción con la potencia de las representaciones de primer orden, concentrándose en particular en las representaciones de teorías como programas lógicos[4]. Tiene popularidad por tres razones. Primero, ILP ofrece un enfoque riguroso al problema general de aprendizaje inductivo basado en el conocimiento. Segundo, ofrece algoritmos completos para inducir teorías de primer orden generales, a partir de los ejemplos. Estos algoritmos aprenden con éxito en dominios donde los algoritmos basados en los atributos son difíciles de aplicar. Un ejemplo consiste en aprender cómo se pliega la estructura de las proteínas (Figura 19.10). La configuración tridimensional de la molécula de las proteínas no puede ser representada razonablemente a través de un conjunto de atributos, ya que la configuración inherente se refiere a las *relaciones* entre los objetos, no a los atributos de un único objeto. La lógica de primer orden es un lenguaje apropiado para describir estas relaciones. Tercero, la programación lógica inductiva produce hipótesis que son (relativamente) fáciles de entender por humanos. Por ejemplo, la traducción al castellano de la Figura 19.10 puede ser examinada y criticada por gente que trabaje en biología. Esto significa que los sistemas de programación lógica inductiva pueden participar en el ciclo científico de la experimentación, generación de hipótesis, debates y refutación. Dichas participaciones no podrían ser posibles para sistemas que generan clasificadores de «caja-negra», como las redes de neuronas.

Un ejemplo

Recuerde de la Ecuación (19.5) que el problema general de inducción basado en el conocimiento es «resolver» la restricción

$$ConocimientoBásico \wedge Hipótesis \wedge Descripciones \models Clasificaciones$$

para la *Hipótesis* desconocida, dados el *ConocimientoBásico* y los ejemplos descritos por las *Descripciones* y las *Clasificaciones*. Para ilustrar esto, usaremos el problema de aprendizaje de relaciones familiares a partir de ejemplos. Las descripciones consistirán en un árbol genealógico extendido, descrito en términos de *Madre*, *Padr*e, la relación

[4] Sería apropiado en este punto para el lector remitirse al Capítulo 9 para algunos de los conceptos que aparecen, incluyendo cláusulas de Horn, forma normal conjuntiva, unificación y resolución.

Figura 19.10 (a) y (b) muestran ejemplos positivos y negativos, respectivamente, del concepto «haz cuatro-hélices arriba-y-abajo» en el dominio del pliegue de proteínas. Cada estructura del ejemplo se codifica como una expresión lógica de unas 100 conjunciones tales como *LongitudTotal*(*D2mhr*, 118)∧*NúmeroHélices*(*D2mhr*, 6) ∧... A partir de estas descripciones y clasificaciones de la forma *Pliegue*(HAZ-CUATRO-HÉLICES ARRIBA-Y-ABAJO, *D2mhr*), el sistema de programación lógica inductiva PROGOL (Muggleton, 1995) aprendía la siguiente regla:

$$\begin{aligned} &Pliegue(\text{HAZ-CUATRO-HÉLICES ARRIBA-Y-ABAJO}, p) \Leftarrow \\ &\quad Hélice(p, h_1) \wedge Longitud(h_1, \text{ALTA}) \wedge Posición(p, h_1, n) \\ &\quad \wedge (1 \leq n \leq 3) \wedge Adyacentes(p, h_1, h_2) \wedge Hélice(p, h_2) \end{aligned}$$

Este tipo de regla podría no ser aprendida, o incluso representada, por un mecanismo basado en los atributos como vimos en los previos capítulos. La regla puede traducirse al castellano como

> La proteína p tiene un pliegue de la clase «haz cuatro-hélices arriba-y-abajo» si contiene una hélice larga h_1 en una posición de estructura secundaria entre 1 y 3 y h_1 está próxima a una segunda hélice.

CasadoCon y las propiedades *Hombre* y *Mujer*. Como ejemplo, usaremos el árbol genealógico del Ejercicio 8.11, mostrado en la Figura 19.11. A continuación se muestran las correspondientes descripciones:

Padre(*Philip*, *Charles*)	*Padre*(*Philip*, *Anne*)	...
Madre(*Mum*, *Margaret*)	*Madre*(*Mum*, *Elizabeth*)	...
CasadoCon(*Diana*, *Charles*)	*CasadoCon*(*Elizabeth*, *Philip*)	...
Hombre(*Philip*)	*Hombre*(*Charles*)	...
Mujer(*Beatrice*)	*Mujer*(*Margaret*)	...

Las sentencias en *Clasificaciones* dependen del concepto objetivo que está siendo aprendido. Podríamos querer aprender *Abuelo/a*, *Cuñado*, o *Ancestro*, por ejemplo. Para *Abuelo/a*, el conjunto completo de *Clasificaciones* contiene $20 \times 20 = 400$ conjunciones de la forma

$$\begin{array}{ll} Abuelo/a(Mum, Charles) & Abuelo/a(Elizabeth, Beatrice) \ \ldots \\ \neg Abuelo/a(Mum, Harry) & \neg Abuelo/a(Spencer, Peter) \ \ldots \end{array}$$

Podríamos, por supuesto, aprender a partir de un subconjunto de este conjunto completo.

El objetivo de un programa de aprendizaje inductivo es proponer un conjunto de sentencias para la *Hipótesis*, tal que la restricción se satisfaga. Suponga, por el momento, que el agente no tiene conocimiento básico: *ConocimientoBásico* es vacío. Entonces una posible solución para la *Hipótesis* es la siguiente:

$$\begin{aligned} Abuelo/a(x, y) \Leftrightarrow\ & [\exists z\ Madre(x, z) \wedge Madre(z, y)] \\ \vee\ & [\exists z\ Madre(x, z) \wedge Padre(z, y)] \\ \vee\ & [\exists z\ Padre(x, z) \wedge Madre(z, y)] \\ \vee\ & [\exists z\ Padre(x, z) \wedge Padre(z, y)] \end{aligned}$$

Nótese que un algoritmo de aprendizaje basado en atributos, tal como APRENDIZAJE-ÁRBOL-DECISIÓN, no conseguirá nada a la hora de resolver este problema. Para poder expresar *Abuelo/a* como un atributo (es decir, un predicado unario), necesitamos convertir pares de personas en objetos:

$$Abuelo/a(\langle Mum, Charles\rangle) \ \ldots$$

Luego nos atascaremos intentando representar las descripciones del ejemplo. Los únicos atributos posibles tienen un aspecto tan horroroso como

$$PrimerElementoEsMadreDeElisabeth(\langle Mum, Charles\rangle)$$

La definición de *Abuelo/a* en términos de estos atributos simplemente se convierte en una gran disyunción de casos específicos que no generaliza para nuevos ejemplos. *Los algoritmos de aprendizaje basados en atributos son incapaces de aprender predicados relacionales*. Por ello, una de las principales ventajas de los algoritmos de ILP es su aplicabilidad a un mayor rango de problemas, incluyendo problemas de relaciones.

Figura 19.11 Un árbol genealógico típico.

Desde luego, el lector habrá notado que un poco de conocimiento básico ayudaría en la representación de la definición de *Abuelo/a*. Por ejemplo, si *ConocimientoBásico* incluye la sentencia

$$Progenitor(x, y) \Leftrightarrow [Madre(x , y) \vee Padre(x, y)]$$

entonces la definición de *Abuelo/a* se reduciría a

$$Abuelo/a(x, y) \Leftrightarrow [\exists z\ Progenitor(x, z) \wedge Progenitor(z, y)]$$

Esto muestra cómo el conocimiento básico puede reducir considerablemente el tamaño de la hipótesis requerido para explicar las observaciones.

Para los algoritmos de ILP también es posible *crear* nuevos predicados para expresar más fácilmente la hipótesis explicativa. Dado el ejemplo de datos mostrado anteriormente, es razonable para el programa ILP proponer un predicado adicional, al que llamaremos «*Progenitor*», para simplificar las definiciones de los predicados meta. Los algoritmos que pueden generar nuevos predicados se denominan algoritmos de **inducción constructiva**. De forma más clara, la inducción constructiva es una parte necesaria de la descripción del aprendizaje acumulativo esbozada en la introducción. Se ha considerado como uno de los problemas más duros en el aprendizaje automático, pero algunas técnicas de ILP proporcionan mecanismos efectivos para conseguirla.

INDUCCIÓN CONSTRUCTIVA

En el resto del capítulo, estudiaremos dos enfoques principales de ILP. El primero usa una generalización de los métodos de árboles de decisión, y el segundo usa técnicas basadas en la inversión de la prueba de resolución.

Métodos de aprendizaje inductivo de arriba a abajo (*Top-down*)

Los primeros enfoques en trabajos de ILP comienzan con una regla muy general que gradualmente se especializa hasta que encaja con los datos. Esto es esencialmente lo que ocurre en el aprendizaje de árboles de decisión, donde se construye incrementalmente un árbol de decisión hasta que es consistente con todas las observaciones. Para ILP usamos literales de primer orden en vez de atributos, y la hipótesis es un conjunto de cláusulas en vez de un árbol de decisión. En esta sección se describe FOIL (Quinlan, 1990), uno de los primeros programas de ILP.

Suponga que estamos intentando aprender la definición de *Abuelo*(*x*,*y*), usando los mismos datos familiares que antes. Al igual que el aprendizaje de árboles de decisión, podemos dividir los ejemplos en positivos y negativos. Los ejemplos positivos son

$$\langle George, Anne\rangle, \langle Philip, Peter\rangle, \langle Spencer, Harry\rangle, \ldots$$

y los ejemplos negativos son

$$\langle George, Elizabeth\rangle, \langle Harry, Zara\rangle, \langle Charles, Philip\rangle, \ldots$$

Nótese que cada ejemplo es un par de objetos, ya que *Abuelo* es un predicado binario. Existen 12 ejemplos positivos en el árbol genealógico y 388 ejemplos negativos (todos ellos pares de personas).

FOIL construye un conjunto de cláusulas, cada una con *Abuelo*(x,y) en la cabeza. Las cláusulas deben clasificar los 12 ejemplos positivos como instancias de la relación *Abuelo*(x,y), y desechar los 388 ejemplos negativos. Las cláusulas son cláusulas de Horn, extendidas con literales negados usando la negación como fallo, al igual que en Prolog. La cláusula inicial tiene cuerpo vacío:

$$\Rightarrow Abuelo(x, y)$$

Esta cláusula clasifica todos los ejemplos como positivos, así que necesita ser especializada. Hacemos esto añadiendo literales a su parte izquierda, de uno en uno. A continuación se muestran tres adiciones posibles:

$$Padre(x, y) \Rightarrow Abuelo(x, y)$$
$$Progenitor(x, z) \Rightarrow Abuelo(x, y)$$
$$Padre(x, z) \Rightarrow Abuelo(x, y)$$

(Nótese que estamos asumiendo que una cláusula que define *Progenitor* forma parte del conocimiento básico.) La primera de estas tres cláusulas clasifica incorrectamente todos los 12 ejemplos como negativos, así que puede ser ignorada. La segunda y la tercera clasifican correctamente los ejemplos positivos, pero la segunda es incorrecta con una gran parte de los ejemplos negativos (dos veces como mucho, ya que permite tanto madres como padres). Luego, preferiremos la tercera cláusula.

Ahora, necesitamos especializar más la cláusula, para desechar los casos en los que x es el padre de z, pero z no es el progenitor de y. Añadiendo el literal *Progenitor*(z,y) obtenemos

$$Padre(x, z) \wedge Progenitor(z, y) \Rightarrow Abuelo(x, y)$$

que clasifica correctamente todos los ejemplos. FOIL encontrará y elegirá este literal, resolviendo la tarea de aprendizaje. En general, FOIL tendrá que buscar en muchas cláusulas sin éxito antes de encontrar la solución correcta.

Este ejemplo es una ilustración sencilla de cómo opera FOIL. En la Figura 19.12 se muestra un esqueleto del algoritmo completo. Esencialmente, el algoritmo construye repetidamente una cláusula, literal a literal, hasta que verifica todo el subconjunto de ejemplos positivos y ninguno de los ejemplos negativos. Entonces los ejemplos positivos cubiertos por la cláusula se borran del conjunto de entrenamiento, y el proceso continúa hasta que no quedan ejemplos positivos restantes. Las dos subrutinas principales que se explican son NUEVOS-LITERALES, que construye todos los posibles nuevos literales para añadir a la cláusula, y ELEGIR-LITERAL, que selecciona el literal que se añade.

NUEVOS-LITERALES toma una cláusula y construye todos los posibles literales «útiles» que pudieran ser añadidos a la cláusula. Usemos como ejemplo la cláusula

$$Padre(x, z) \Rightarrow Abuelo(x, y)$$

Se pueden añadir tres clases de literales:

1. *Literales que utilizan predicados*: el literal puede estar negado o no, se puede usar cualquier predicado existente (incluyendo el predicado meta), y todos los argumentos deben ser variables. Se puede usar cualquier variable en cualquier argumento del predicado, con una restricción: cada literal debe incluir *al menos una* variable de un literal anterior o de la cabeza de la cláusula. Se permiten li-

```
función FOIL(ejemplos, objetivo) devuelve un conjunto de cláusulas de Horn
   entradas: ejemplos, conjunto de ejemplos
             objetivo, un literal para el predicado meta
   variables locales: cláusulas, conjunto de cláusulas, inicialmente vacío

   mientras ejemplos contenga ejemplos positivos hacer
      cláusula ← NUEVA-CLÁUSULA(ejemplos, objetivo)
      borrar los ejemplos cubiertos por cláusula en ejemplos
      añadir cláusula a cláusulas
   devolver cláusulas
```

```
función NUEVA-CLÁUSULA(ejemplos, objetivo) devuelve una cláusula de Horn
   variables locales: cláusula, una cláusula con objetivo como cabeza y con cuerpo vacío
                      l, un literal para añadirse a la cláusula
                      ejemplos_extendidos, un conjunto de ejemplos con valores para
                         nuevas variables

   ejemplos_extendidos ← ejemplos
   mientras ejemplos_extendidos contenga ejemplos negativos hacer
      l ← ELEGIR-LITERAL(NUEVOS-LITERALES(cláusula), ejemplos_extendidos)
      añadir l al cuerpo de la cláusula
      ejemplos_extendidos ← conjunto de ejemplos creados aplicando
         EXTENDER-EJEMPLO a cada ejemplo en ejemplos_extendidos
   devolver cláusula
```

```
función EXTENDER-EJEMPLO (ejemplo, literal) devuelve
   si ejemplo satisface el literal
      entonces devolver el conjunto de ejemplos creados extendiendo ejemplo con
         cada posible valor de constante para cada nueva variable en literal
   si no devolver el conjunto vacío
```

Figura 19.12 Esqueleto del algoritmo FOIL para aprendizaje de conjuntos de cláusulas de Horn de primer orden a partir de ejemplos. NUEVOS-LITERALES y ELEGIR-LITERAL se explican en el texto.

terales tales como $Madre(z, u)$, $CasadoCon(z, z)$, $\neg Hombre(y)$ y $Abuelo(v, x)$, mientras que $CasadoCon(u, v)$ no. Nótese que el uso del predicado de la cabeza de la cláusula permite a FOIL el aprendizaje de definiciones *recursivas*.

2. *Literales de igualdad y no-igualdad*: relacionan variables que ya han aparecido en la cláusula. Por ejemplo, podemos añadir $z \neq x$. Estos literales también pueden incluir constantes especificadas por el usuario. Para el aprendizaje de aritmética podríamos usar el 0 y el 1, y para el aprendizaje de listas de funciones podríamos usar la lista vacía [ ].
3. *Comparaciones aritméticas*: cuando se trata con funciones de variables continuas, se pueden añadir literales de la forma $x > y$ y $y \leq z$. Al igual que en el aprendizaje de árboles de decisión se puede elegir un valor constante que sirva de umbral para maximizar el poder discriminatorio del test.

El factor de expansión resultante en este espacio de búsqueda es muy grande (*véase* el Ejercicio 19.6), pero FOIL puede usar información de tipo para reducirlo. Por ejemplo,

si el dominio incluye números y personas, las restricciones de tipo prevendrían que NUEVOS-LITERALES generase literales de la forma *Padre*(*x*,*n*), donde *x* es una persona y *n* es un número.

ELEGIR-LITERAL usa una heurística parecida a la ganancia de información (*véase* Apartado 18.3) para decidir qué literal añadir. Los detalles exactos no son importantes aquí, pero se ha probado con distintas variantes. Una característica adicional de FOIL es el uso de la navaja de Ockham para eliminar algunas hipótesis. Si una cláusula se hace más larga (de acuerdo con alguna métrica) que la longitud total de los ejemplos positivos que la cláusula explica, dicha cláusula no se considera como una hipótesis potencial. Esta técnica proporciona una forma de evitar cláusulas demasiado complejas que producen ruido en los datos. Consultar el Apartado 20.1 para una explicación de la relación entre el ruido y el tamaño de las cláusulas.

FOIL y sus semejantes se han usado para aprender una amplia variedad de definiciones. Una de las demostraciones más impresionantes (Quinlan y Cameron-Jones, 1993) suponía resolver una gran secuencia de ejercicios de funciones de procesamiento de listas del libro de Prolog de Bratko (1986). En cada caso, el programa era capaz de aprender una definición correcta de la función a partir de un pequeño conjunto de ejemplos, usando las funciones aprendidas previamente como conocimiento básico.

Aprendizaje inductivo con deducción inversa

RESOLUCIÓN INVERSA

El segundo enfoque principal de ILP supone invertir el proceso normal de prueba de deducción. La **resolución inversa** se basa en la observación de que si el ejemplo *Clasificaciones* resulta de *ConocimientoBásico* $\wedge$ *Hipótesis* $\wedge$ *Restricciones*, entonces se puede probar este hecho a través de resolución (ya que la resolución es completa). Si podemos «ejecutar la prueba hacia atrás», podemos encontrar una hipótesis tal que la prueba llegue a ella. La clave es encontrar una forma de invertir el proceso de resolución.

Mostraremos un proceso de prueba hacia atrás para la resolución inversa que consiste en pasos hacia atrás individuales. Recordando que un proceso normal de resolución toma dos cláusulas C_1 y C_2 y las resuelve produciendo el **resolvente** C. Un paso de resolución inversa toma un resolvente C y produce dos cláusulas C_1 y C_2, tal que C es el resultado de resolver C_1 y C_2. De forma alternativa, se podría tomar un resolvente C y una cláusula C_1 y producir una cláusula C_2 de forma que C es el resultado de resolver C_1 y C_2.

En la Figura 19.13 se muestran los primeros pasos del proceso de resolución inversa, donde nos centramos en el ejemplo positivo *Abuelo*/*a*(*George*, *Anne*). El proceso comienza al final de la prueba (mostrada en la parte inferior de la figura). Tomamos el resolvente C como la cláusula vacía (es decir, una contradicción) y C_2 como $\neg$*Abuelo*/*a*(*George*, *Anne*), que es la negación del ejemplo meta. El primer paso inverso toma C y C_2 y genera la cláusula *Abuelo*/*a*(*George*, *Anne*) para C_1. El siguiente paso toma esta cláusula como C y la cláusula *Progenitor*(*Elizabeth*, *Anne*) como C_2, y genera la cláusula

$$\neg Progenitor(Elizabeth, y) \vee Abuelo/a(George,y)$$

como C_1. El último paso trata esta cláusula como resolvente. Con *Progenitor*(*George*, *Elizabeth*) como C_2, una posible cláusula C_1 es la hipótesis

$$Progenitor(x, z) \wedge Progenitor(z, y) \Rightarrow Progenitor/a(x, y)$$

Ahora tenemos una prueba de resolución que a partir de las hipótesis, descripciones y el conocimiento básico se deduce la clasificación *Abuelo*/*a*(*George*, *Anne*).

Obviamente, la resolución inversa requiere una búsqueda. Cada paso de resolución inversa es no determinístico, ya que para cualquier C, puede haber muchos o incluso un número infinito de cláusulas C_1, C_2 que resuelvan C. Por ejemplo, en vez de elegir $\neg Progenitor(Elizabeth, y) \vee Abuelo/a(George, y)$ como C_1 en el último paso de la Figura 19.13, el paso de resolución inversa podría haber elegido cualquiera de las siguientes sentencias:

$$\neg Progenitor(Elizabeth, Anne) \vee Abuelo/a(George, Anne)$$
$$\neg Progenitor(z, Anne) \vee Abuelo/a(George, Anne)$$
$$\neg Progenitor(x, y) \vee Abuelo/a(George, y)$$
$$\vdots$$

(*Véanse* los Ejercicios 19.4 y 19.5.) Además, las cláusulas que participan en cada paso pueden elegirse del *ConocimientoBásico*, de la *Descripción* de los ejemplos, de la negación de las *Clasificaciones*, o de las cláusulas hipótesis que ya han sido generadas en el árbol de resolución inversa. El gran número de posibilidades conlleva un gran factor de ramificación (y por ello una búsqueda ineficiente) sin controles adicionales. En sistemas ILP se han implementado varios enfoques para mejorar la búsqueda:

1. Pueden eliminarse opciones redundantes, por ejemplo, generando sólo la hipótesis más específica posible y requiriendo que todas las cláusulas hipótesis sean consistentes entre sí, y con las observaciones. Este último criterio desecharía la cláusula $\neg Progenitor(z, y) \vee Abuelo/a(George, y)$, mostrada anteriormente.

Figura 19.13 Primeros pasos en el proceso de resolución inversa. Las cláusulas sombreadas se generan a través de los pasos de resolución inversa a partir de la cláusula de la derecha y de la cláusula de abajo. Las cláusulas sin sombrear provienen de *Descripciones* y *Clasificaciones*.

2. Se puede restringir la estrategia de prueba. Por ejemplo, en el Capítulo 9 se mostró que la **resolución lineal** es completa. Las estrategias reducidas permiten que los árboles de prueba tengan sólo estructuras de ramificación lineal (tal como en la Figura 19.13).
3. Se puede restringir el lenguaje de representación, por ejemplo eliminando los símbolos de funciones o permitiendo sólo cláusulas de Horn. Por ejemplo, PROGOL opera con cláusulas de Horn usando **deducción inversa**. La idea es cambiar la restricción

$$ConocimientoBásico \wedge Hipótesis \wedge Descripciones \models Clasificaciones$$

por la siguiente fórmula lógicamente equivalente

$$ConocimientoBásico \wedge Descripciones \wedge \neg Clasificaciones \models \neg Hipótesis$$

DEDUCCIÓN INVERSA

A partir de esto, se puede usar un proceso similar a la deducción normal con la negación como fallo que realiza Prolog con cláusulas de Horn, para derivar las *Hipótesis*. Ya que se restringe a cláusulas de Horn, es un método incompleto, pero puede ser más eficiente que la resolución completa. También es posible aplicar inferencia completa con deducción inversa, Inoue (2001).
4. Se puede hacer la inferencia con comprobación de modelos en vez de con demostración de teoremas. El sistema PROGOL (Muggleton, 1995) usa un tipo de comprobación de modelos para limitar la búsqueda. Es decir, al igual que en la programación de conjuntos, genera posibles valores para variables lógicas, y comprueba la consistencia.
5. Se puede hacer la inferencia con cláusulas proposicionales instanciadas en vez de con lógica de primer orden. El sistema LINUS (Lavrač y Džeroski, 1994) funciona traduciendo teorías en lógica de primer orden a lógica proposicional, resolviéndolas con un sistema de aprendizaje proposicional, y luego volviéndolas a traducir. Trabajar con fórmulas proposicionales puede ser más eficiente en algunos problemas, como se mostró con SATPLAN en el Capítulo 11.

Hacer descubrimientos con la programación lógica inductiva

Un procedimiento de resolución inversa que invierte completamente la estrategia de resolución es, en principio, un algoritmo completo para el aprendizaje de teorías de primer orden. Es decir, si alguna *Hipótesis* desconocida genera un conjunto de ejemplos, un procedimiento de resolución inversa puede generar *Hipótesis* a partir de ejemplos. Esta observación sugiere una posibilidad interesante: suponga que los ejemplos disponibles incluyen varias trayectorias de cuerpos cayendo. ¿Podría un programa de resolución inversa ser teóricamente capaz de inferir la ley de la gravedad? Claramente la respuesta es que sí, ya que la ley de la gravedad permite explicar los ejemplos, dada una base matemática adecuada. De forma similar, podemos imaginar que el electromagnetismo, la mecánica cuántica y la teoría de la relatividad también pertenecen al ámbito de los programas de ILP. Por supuesto, se puede decir que también están al alcance de

un mono con una máquina de escribir; todavía necesitamos heurísticas mejores y nuevas formas de estructurar el espacio de búsqueda.

Algo que los sistemas de resolución inversa *pueden* hacer por nosotros es inventar nuevos predicados. Esta capacidad a menudo parece algo mágica, ya que normalmente parece que los computadores «trabajan sólo con lo que se les da». De hecho, los nuevos predicados se generan directamente a partir de los pasos de resolución inversa. El caso más simple aparece cuando tomamos como hipótesis dos nuevas cláusulas C_1 y C_2, dada una cláusula C. La resolución de C_1 y C_2 elimina un literal que las dos cláusulas comparten; a partir de aquí, es bastante posible que el literal eliminado contenga un predicado que no aparezca en C. Por ello, cuando se trabaja hacia atrás, una posibilidad es generar un nuevo predicado a partir de reconstruir el literal que falta.

La Figura 19.14 muestra un ejemplo en el cual el nuevo predicado P se genera en el proceso de aprendizaje de la definición de *Antecesor*. Una vez generado, P se puede usar en pasos posteriores de resolución inversa. Por ejemplo, un paso posterior podría tomar como hipótesis $Madre(x, y) \Rightarrow P(x, y)$. Por ello, el nuevo predicado P tiene su significado restringido por la generación de hipótesis relacionadas con él. Otro ejemplo podría llevarnos a la restricción $Padre(x, y) \Rightarrow P(x, y)$. En otras palabras, el predicado P es lo que hemos denominado la relación *Progenitor*. Como mencionamos anteriormente, la invención de nuevos predicados puede reducir el tamaño de la definición del predicado meta. Por lo tanto, incluyendo la habilidad de inventar nuevos predicados, los sistemas de resolución inversa pueden resolver a menudo problemas que es inviable resolver con otras técnicas.

Algunas de las revoluciones más trascendentes que se han producido en la ciencia vienen de la invención de nuevos predicados y funciones, por ejemplo, la invención de la aceleración de Galileo, o la invención de la energía térmica de Joule. Una vez que estos términos están disponibles, el descubrimiento de nuevas leyes se convierte en una tarea (relativamente) fácil. La parte difícil recae en darse cuenta de que alguna nueva entidad, con una relación específica con entidades existentes, permitiría que un cuerpo entero de observaciones sea explicado de una forma más simple y con teorías más elegantes de las que previamente existían.

Hasta ahora, los sistemas ILP no han hecho descubrimientos al nivel de Galileo o Joule, pero sus descubrimientos han sido juzgados como publicables en la literatura científica. Por ejemplo, en la revista *Journal of Molecular Biology*, Turcotte *et al.* (2001) describen el descubrimiento automático de reglas para el pliegue de proteínas usando el programa PROGOL de ILP. Muchas de las reglas descubiertas por PROGOL podrían haber

Figura 19.14 Un paso de resolución inversa que genera un nuevo predicado P.

sido derivadas a partir de principios conocidos, pero la mayoría no habían sido previamente publicadas como parte de una base de datos biológicos estándar. (Puede observar un ejemplo en la Figura 19.10.) En trabajos relacionados, Srinivasan *et al*. (1994) trataban el problema de descubrir reglas basadas en la estructura molecular para la mutación de componentes nitroaromáticas. Estas componentes se encontraron en gases de automóviles. Para el 80 por ciento de las componentes de una base de datos estándar, es posible identificar cuatro características importantes. La regresión lineal sobre estas características mejora a ILP. Para el restante 20 por ciento, las características por separado no pueden predecirse, e ILP identifica relaciones que le permiten mejorar la regresión lineal, las redes de neuronas y los árboles de decisión. King *et al*. (1992) mostró cómo predecir la eficacia terapéutica de varias drogas a partir de su estructura molecular. En todos estos ejemplos se observa que tanto la habilidad de representar relaciones como el uso de conocimiento básico contribuyen al alto rendimiento de ILP. El hecho de que las reglas encontradas por ILP puedan ser interpretadas por los humanos contribuye a la aceptación de estas técnicas tanto en revistas de Biología como en revistas de informática.

ILP ha contribuido a otras ciencias además de la Biología. Una de las más importantes es el procesamiento del lenguaje natural, donde ILP se ha usado para extraer información relacional compleja a partir de un texto. Estos resultados se resumen en el Capítulo 23.

18.6 Resumen

En este capítulo se han investigado varias formas de utilizar el conocimiento *a priori* para ayudar a un agente a aprender de nuevas experiencias. Ya que mucho conocimiento *a priori* se expresa más en términos de modelos relacionales que en modelos basados en atributos, el capítulo también cubre sistemas que permiten el aprendizaje de modelos relacionales. Los puntos importantes son:

- El uso de conocimiento *a priori* en el aprendizaje lleva al **aprendizaje acumulativo**, en el que el agente mejora su habilidad de aprender a medida que adquiere más conocimiento.
- El conocimiento *a priori* ayuda en el aprendizaje mediante la eliminación de hipótesis, en otro caso consistente y «rellenando» la explicación de ejemplos, por consiguiente, permitiendo hipótesis más cortas. Estas contribuciones frecuentemente dan como resultado un aprendizaje más rápido a partir de menos ejemplos.
- Entender los diferentes roles lógicos que juega el conocimiento *a priori*, expresados por **restricciones**, ayuda a definir una variedad de técnicas de aprendizaje.
- **El aprendizaje basado en explicaciones** (EBL) extrae reglas generales a partir de ejemplos simples, *explicando* los ejemplos y generalizando la explicación. Proporciona un método deductivo que convierte los primeros principios de conocimiento en habilidades de propósito especial útiles y eficientes.
- **El aprendizaje basado en relevancia** (RBL) utiliza el conocimiento *a priori* en forma de determinaciones para identificar atributos relevantes, generando un es-

pacio de hipótesis reducido y acelerando así el aprendizaje. El RBL también permite generalizaciones deductivas a partir de ejemplos simples.

- **El aprendizaje inductivo basado en el conocimiento** (KBIL) encuentra hipótesis inductivas que explican conjuntos de observaciones con la ayuda de conocimiento básico.
- Las técnicas de **programación lógica inductiva** (ILP) utilizan KBIL sobre conocimiento expresado en lógica de primer orden. Los métodos ILP pueden aprender conocimiento relacional que no se puede expresar en los sistemas basados en atributos.
- Se puede llevar a cabo ILP mediante un enfoque de arriba a abajo (*top-down*) de refinamiento de una regla general, o mediante un enfoque de abajo a arriba (*bottom-up*) de inversión del proceso deductivo.
- Los métodos ILP generan, de forma natural, nuevos predicados con los que se pueden expresar nuevas teorías de una forma concisa, y son muy prometedores como sistemas de formación de teorías científicas de propósito general.

Notas bibliográficas e históricas

Aunque el uso de conocimiento *a priori* en al aprendizaje era algo natural para los filósofos de la ciencia, no se ha realizado mucho trabajo formal sobre ello hasta hace poco. *Fact, Fiction and Forecast* (Hechos, Ficción y Predicción), del filósofo Nelson Goodman (1954), refuta las suposiciones iniciales de que la inducción simplemente se ocupa de observar suficientes ejemplos de alguna proposición cuantificada universalmente para después adoptarla como hipótesis. Considere, por ejemplo, la hipótesis «Todas las esmeraldas son verdeazules», donde **verdeazul** significa «verde si se observa antes de un tiempo *t*, pero azul si se observa después». En cualquier instante anterior a *t*, podríamos haber observado millones de instancias que confirman la regla de que las esmeraldas son verdeazules, y ninguna instancia que desconfirme la regla, y aún ser incapaces de adoptar la regla. Esto se explica únicamente por el papel que juega el conocimiento *a priori* relevante en el proceso de inducción. Goodman distingue distintos tipos útiles de conocimiento *a priori*, incluyendo una versión de las determinaciones denominada **sobrehipótesis**. Desafortunadamente, las ideas de Goodman nunca se aplicaron en aprendizaje automático.

VERDEAZUL

Las raíces del EBL vienen de las técnicas utilizadas en el planificador STRIPS (Fikes *et al*. 1972). Cuando se construye un plan, se guarda una versión generalizada del mismo en una librería de planes, que se utiliza en planificaciones posteriores como **macro-operador**. Ideas similares a ésta aparecían en la arquitectura ACT* de Anderson, bajo el título de **compilación de conocimiento** (*knowledge compilation*) (Anderson, 1983), y en la arquitectura SOAR, como **fragmentación** (*chunking*) (Laird *et al*., 1986). La **adquisición de esquemas** (DeJong, 1981), la **generalización analítica** (Mitchell, 1982) y la **generalización basada en restricciones** (Minton, 1984) son consecuencias inmediatas del rápido crecimiento del interés en EBL, estimulado por los artículos de Mitchell *et al*. (1986) y de DeJong y Mooney (1986). El algoritmo de EBL descrito en el texto, fue descrito por Hirsh (1987), que mostraba cómo se podría incorporar direc-

tamente en un sistema de programación lógica. Van Harmelen y Bundy (1988) explican EBL como una variante del método de **evaluación parcial** que se utiliza en los sistemas de análisis de programas (Jones *et al*., 1993.)

Más recientemente, los análisis rigurosos realizados han conducido a una mejor comprensión de los costes potenciales y beneficios del EBL en términos de velocidad en la resolución de problemas. Minton (1988) ha mostrado que con un trabajo extra extensivo, el EBL podría fácilmente reducir la velocidad de un programa de forma significativa. Tambe *et al*. (1990) también encontraron un problema similar con el mecanismo de fragmentación (*chunking*), y proponían una reducción en la potencia expresiva del lenguaje de reglas para minimizar el coste de equiparación de las reglas con la memoria de trabajo. Este trabajo tiene mucha relación con los resultados que se han obtenido recientemente sobre la complejidad de la inferencia en versiones restringidas de la lógica de primer orden. (*Véase* Capítulo 9.) Se puede encontrar un análisis probabilístico formal de el coste esperado en EBL en Greiner (1989) y Subramanian y Feldman (1990). En Dietterich (1990) hay un estudio excelente.

RAZONAMIENTO ANALÓGICO

En vez de utilizar los ejemplos como punto de partida para la generalización, también se pueden utilizar directamente para resolver nuevos problemas, mediante un proceso conocido como **razonamiento analógico**. Esta forma de razonamiento se puede llevar a cabo realizando un tipo de razonamiento plausible basado en grados de similaridad (Gentner, 1983), realizando un tipo de inferencia deductiva basada en determinaciones que utiliza el ejemplo (Davies y Russell, 1987), o realizando un tipo de EBL «perezoso» que dirige la generalización del ejemplo antiguo para que se adecue a las necesidades del nuevo problema. Es común que esta última forma de razonamiento analógico se realice en el **razonamiento basado en casos** (*case-based reasoning*) (Kolodner, 1993) y en la **analogía derivacional** (Veloso y Carbonell, 1993).

La relevancia de la información en forma de dependencias funcionales fue inicialmente desarrollada en la comunidad de bases de datos, donde se utiliza para estructurar conjuntos de atributos grandes en forma de subconjuntos manejables. Las dependencias funcionales fueron utilizadas por Carbonell y Collins (1973). Bobrow y Raphael (1974) las dotaron de un aire más lógico. Las dependencias fueron redescubiertas de forma independiente por Davies y Russell (Davies, 1985; Davies y Russell, 1987), que proporcionaron un análisis lógico completo. Fueron utilizadas para restricciones declarativas por Russell y Grosof (1987). Russell (1988) demostró la equivalencia entre las determinaciones y un vocabulario restringido del espacio de hipótesis. El algoritmo FOCUS, en Almuallim y Dietterich (1991), fue el primer algoritmo de aprendizaje para determinaciones que mostraba la mejora del rendimiento que se obtiene con RBDTL. En Tadepalli (1993) se describe un algoritmo ingenioso para aprendizaje de determinaciones que presenta muchas mejoras en la velocidad del aprendizaje.

La idea de que el aprendizaje inductivo se puede realizar mediante deducción inversa se debe a W.S. Jevons (1874), que escribió: «El estudio de la Lógica Formal y de la Teoría de las Probabilidades me ha llevado a adoptar la opinión de que no hay nada tan diferente como los métodos de deducción e inducción, pero la inducción es simplemente un uso inverso de la deducción». Las investigaciones computacionales comenzaron con la tesis de Gordon Plotkin (1971) en Edimburgo. Aunque Plotkin desarrolló muchos de los teoremas y métodos que hoy en día se utilizan en ILP, se desanimó a causa de algu-

nos resultados de indecibilidad en ciertos problemas de inducción. MIS (Shapiro, 1981) volvió a introducir el problema de aprendizaje de programas lógicos, pero principalmente se consideró como una contribución a la teoría de la depuración automática. Los trabajos en inducción de reglas, como ID3 (Quinlan, 1986) y CN2 (Clark y Noblett, 1989), llevaron a FOIL (Quinlan, 1990), que permite, por primera vez, la inducción de reglas relacionales. El campo del aprendizaje relacional fue de nuevo impulsado por Muggleton y Buntine (1988), cuyo programa CIGOL incorporaba una versión incompleta de resolución inversa que era capaz de generar nuevos predicados[5]. Wirth y O'Rorke (1991) también trabajaron en la generación de nuevos predicados. El siguiente sistema fue GOLEM (Muggleton y Feng, 1990), que utiliza un algoritmo de cubrimiento basado en el concepto de Plotkin de generalización mínima relativa. Mientras que FOIL trabaja de arriba abajo (*top-down*), CIGOL y GOLEM lo hacen de abajo a arriba (*botton-up*). Otros sistemas de la misma época son ITOU (Rouveirol y Puget, 1989) y CLINT (De Raedt, 1992). Más recientemente, ha aparecido PROGOL (Muggleton, 1995), que utiliza un enfoque híbrido (de arriba abajo y de abajo a arriba) para realizar deducción inversa, y que ha sido aplicado a varios problemas prácticos, especialmente de Biología y procesamiento de lenguaje natural. Muggleton (2000) describe una extensión de PROGOL para manejar incertidumbre en forma de programas lógicos estocásticos.

En Muggleton (1991), y en un gran número de artículos de Muggleton (1992), se puede encontrar un análisis formal de los métodos de ILP. En el libro de Lavrač y Džeroski (1994) se puede encontrar una colección de técnicas y aplicaciones. Page y Srinivasan (2002) proporcionan una visión general más reciente de la historia del campo y los retos futuros. Los primeros resultados relativos a la complejidad, debidos a Haussler (1989), sugirieron que aprender sentencias de lógica de primer orden era muy complejo. Sin embargo, con un mejor entendimiento de la importancia de los distintos tipos de restricciones sobre las cláusulas, se obtuvieron resultados más positivos, incluso para cláusulas con recursividad (Dzeroski *et al.* 1992). Los resultados sobre la capacidad de aprendizaje de ILP fueron estudiados por Kietz y Dzeroski (1994) y Cohen y Page (1995).

SISTEMAS DE DESCUBRIMIENTO

Aunque hoy en día parece que ILP es el enfoque dominante para inducción constructiva, no es el único. Los denominados **sistemas de descubrimiento** (*discovery systems*) tienen por objetivo la modelización del proceso que se lleva a cabo para el descubrimiento científico de nuevos conceptos, que normalmente se realiza mediante una búsqueda directa en el espacio de las definiciones de conceptos. El *Automated Mathematician*, o AM, de Doug Lenat (Davis y Lenat, 1982) utilizaba heurísticas de descubrimiento expresadas como reglas de un sistema experto para guiar su búsqueda de conceptos y conjeturas en la teoría elemental de números. Al contrario que la mayoría de los sistemas diseñados para razonamiento matemático, AM carecía del concepto de prueba, y sólo podía hacer conjeturas. Redescubrió la conjetura de Goldbach y el teorema de Factorización Única de Números Primos. La arquitectura de AM fue generalizada en el sistema EURISKO (Lenat, 1983) añadiendo un mecanismo capaz de reescribir las heurísticas de descubrimiento propias del sistema, aunque con menos éxito que AM.

[5] El método de resolución inversa también aparece en (Russell, 1986), con un algoritmo simple que se proporciona en una nota al pie.

La metodología de AM y EURISKO ha sido muy discutida (Ritchie y Hanna, 1984; Lenat y Brown, 1984).

Otros tipos de sistemas de descubrimiento tienen por objetivo operar con datos científicos reales para encontrar nuevas leyes. Los sistemas DALTON, GLAUBER y STAHL (Langley *et al.*, 1987) son sistemas basados en reglas que buscan relaciones cuantitativas en los datos experimentales de sistemas físicos; en cada caso, el sistema ha sido capaz de resumir un descubrimiento bien conocido de la historia de la ciencia. Los sistemas de descubrimiento basados en técnicas probabilísticas (especialmente los algoritmos de agrupamiento que descubren nuevas categorías) se discuten en el Capítulo 20.

EJERCICIOS

19.1 Demuestre, traduciendo a forma normal conjuntiva y aplicando resolución, que la conclusión a la que se llega en el Apartado 19.4, relativa a los brasileños, es completa.

19.2 Para cada una de las determinaciones siguientes, escriba su representación lógica y explique por qué la determinación es cierta (si lo es):

a) El código postal determina el estado en Estados Unidos.
b) El diseño y el valor determinan la masa de una moneda.
c) Para un programa dado, la entrada determina la salida.
d) El clima, la ingestión de comida, el ejercicio y el metabolismo determinan la ganancia y pérdida de peso.
e) La calvicie viene determinada por la calvicie (o la falta de la misma) de uno de los abuelos maternos.

19.3 ¿Sería útil una versión probabilística de las determinaciones? Sugiera una definición.

19.4 Rellene los valores que faltan en las cláusulas C_1 o C_2 (o ambas) en los siguientes conjuntos de cláusulas, dado que C es el resolvente de C_1 y C_2:

a) $C = Verdadero \Rightarrow P(A, B)$, $C_1 = P(x, y) \Rightarrow Q(x, y)$, $C_2 = ??$
b) $C = Verdadero \Rightarrow P(A, B)$, $C_1 = ??$, $C_2 = ??$
c) $C = P(x, y) \Rightarrow P(x, f(y))$, $C_1 = ??$, $C_2 = ??$

Si existiera más de una solución posible, proporcione un ejemplo de cada tipo.

19.5 Suponga que alguien escribe un programa lógico que lleva a cabo un paso de resolución, que es: $Resolver(c_1, c_2, c)$ es cierto si c es el resultado de aplicar resolución a c_1 y a c_2. Normalmente *Resolver* se utilizará como parte de un demostrador de teoremas, haciendo llamadas con c_1 y c_2 instanciadas a cláusulas particulares, y generando así el resolvente c. Ahora suponga que llamamos a *Resolver* con c instanciada, y c_1 y c_2 libres. ¿Tendrá éxito esta llamada en la generación de los resultados apropiados de un paso de resolución inversa? ¿Necesitaríamos realizar modificaciones especiales en un sistema de programación lógica para que esto funcionara?

19.6 Suponga que FOIL está considerando añadir un literal a una cláusula, utilizando un predicado P, y que los literales previos (incluyendo la cabeza de la cláusula) contienen cinco variables diferentes.

a) ¿Cuántos literales funcionalmente diferentes pueden ser generados? Dos literales son funcionalmente idénticos si difieren únicamente en los nombres de las variables *nuevas* que contienen.

b) ¿Podría encontrar una fórmula general para el número de literales diferentes con un predicado de aridad r, cuando se han utilizado previamente n variables?

c) ¿Por qué FOIL no permite literales que contienen variables que previamente no se han utilizado?

19.7 Utilizando los datos del árbol genealógico de la Figura 19.11, o un subconjunto de él, aplique el algoritmo FOIL para aprender una definición del predicado *Antecesor*.

20 Métodos estadísticos de aprendizaje

En este capítulo presentamos el aprendizaje como una forma de razonamiento con incertidumbre a partir de observaciones

En la Parte V mencionamos el predominio de la incertidumbre en los entornos reales. Los agentes pueden tratar con la incertidumbre utilizando métodos que hagan uso de probabilidades y de la teoría de la decisión, pero primero, deben aprender sus teorías probabilísticas sobre el mundo a partir de la experiencia. Este capítulo explica cómo pueden hacer esto. Veremos cómo formular la propia tarea de aprendizaje como un proceso de inferencia probabilística (Apartado 20.1). También veremos que una visión bayesiana del aprendizaje es extremadamente potente, ya que proporciona soluciones generales a los problemas de ruido, sobreajuste y predicción óptima. También tiene en cuenta el hecho de que un agente poco omnisciente nunca estará seguro sobre qué teoría del mundo es correcta, aunque deba tomar decisiones basándose en alguna teoría del mundo.

Se describen métodos para el aprendizaje de modelos probabilísticos (principalmente redes bayesianas) en los Apartados 20.2 y 20.3. El Apartado 20.4 incluye métodos de aprendizaje que almacenan y recuperan instancias específicas. El Apartado 20.5 está dedicado a las redes neuronales y el Apartado 20.6 introduce las máquinas núcleo. Este capítulo incluye material que es bastante matemático (para su completo entendimiento es necesario conocer las bases del cálculo multivariante), aunque las lecciones generales se pueden entender sin entrar en detalles. En este punto, puede ser beneficioso para el lector revisar el material de los capítulos 13, 14, y la base matemática del Apéndice A.

20.1 Aprendizaje estadístico

Los conceptos clave de este capítulo son los **datos** y las **hipótesis**, como en el Capítulo 18. Aquí, los datos son **evidencias,** es decir, instanciaciones de todas o algunas de las

variables aleatorias que describen el dominio. Las hipótesis son teorías probabilísticas sobre cómo funciona el dominio, incluyendo teorías lógicas como caso especial.

Consideremos un ejemplo *muy* simple. Nuestros caramelos sorpresa favoritos son de dos sabores: cereza (¡uhmm!) y lima (¡ughh!). El fabricante de los caramelos tiene un sentido del humor muy peculiar, y envuelve los caramelos en un envoltorio opaco en el que no se indica el sabor. Los caramelos se introducen en grandes bolsas, que son de cinco tipos, otra vez indistinguibles desde fuera:

h_1: 100% cereza
h_2: 75% cereza + 25% lima
h_3: 50% cereza + 50% lima
h_4: 25% cereza + 75% lima
h_5: 100% lima

Dada una nueva bolsa, la variable aleatoria H (para las *hipótesis*) denota el tipo de bolsa, así que puede tomar valores desde h_1 hasta h_5. Por supuesto, H no es directamente observable. Cuando se abren e inspeccionan los caramelos, se revelan los datos D_1, D_2, ..., D_N, donde cada D_i es una variable aleatoria con valores posibles *cereza* y *lima*. La tarea básica a la que se enfrenta el agente es predecir el sabor del siguiente caramelo[1]. A pesar de que aparentemente parece trivial, este escenario sirve para introducir muchos de los aspectos principales. Realmente, el agente necesita inferir una teoría de su mundo, aunque sea muy simple.

APRENDIZAJE BAYESIANO

El **aprendizaje bayesiano** simplemente calcula la probabilidad de cada hipótesis dados los datos, y realiza predicciones sobre estas bases. Es decir, se realizan las predicciones utilizando *todas* las hipótesis, ponderadas por sus probabilidades, y no utilizando únicamente la «mejor» hipótesis. De esta forma, el aprendizaje se reduce a inferencia probabilística. Si **D** representa todos los datos, y **d** el valor observado; la probabilidad de cada hipótesis se obtiene aplicando la regla de Bayes:

$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}|h_i)P(h_i) \tag{20.1}$$

Ahora, suponga que queremos hacer una predicción sobre una cantidad desconocida X. Tenemos

$$\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|\mathbf{d}, h_i)\mathbf{P}(h_i|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)P(h_i|\mathbf{d}) \tag{20.2}$$

donde se ha asumido que cada hipótesis determina una distribución de probabilidades sobre X. Esta ecuación muestra que las predicciones son el resultado de ponderar las predicciones de las hipótesis individuales. Las hipótesis son en sí mismas «intermediarios» entre los datos crudos y las predicciones. Las cantidades clave en el enfoque bayesiano son las **hipótesis a priori**, $P(h_i)$ y la **verosimilitud** de los datos dada cada una de las hipótesis, $P(\mathbf{d}|h_i)$.

HIPÓTESIS A PRIORI

VEROSIMILITUD

En el ejemplo de los caramelos por el momento asumiremos, como información proporcionada por el fabricante, que la distribución a priori sobre h_1, ..., h_5 viene dada por

[1] Los lectores estadísticamente sofisticados reconocerán este escenario como una variante de la **urna y las bolas**. Consideramos este caso menos restringido que el de los caramelos; además el caso de los caramelos se presta a otras tareas, como decidir si cambiar la bolsa con un amigo.

I.I.D

<0,1, 0,2, 0,4, 0,2, 0,1>. La verosimilitud de los datos se calcula asumiendo que las observaciones son **i.i.d.** (es decir, independientes e idénticamente distribuidas) así que

$$P(\mathbf{d}|h_i) = \prod_j P(d_j|h_i) \tag{20.3}$$

Por ejemplo, suponga que la bolsa es una de las que contiene sólo caramelos de lima (h_5), y los primeros 10 caramelos son todos de lima; en este caso, $P(\mathbf{d} \mid h_3)$ es $0{,}5^{10}$, porque la mitad de los caramelos de una bolsa h_3 son de lima[2]. La Figura 20.1(a) muestra cómo cambian las probabilidades a posteriori de las cinco hipótesis a medida que se van observando los 10 caramelos de lima. Nótese que las probabilidades comienzan con sus valores a priori, por lo que h_3 es inicialmente más probable que las demás, incluso después de que se desenvuelva el primer caramelo. Después de desenvolver dos caramelos de lima, h_4 es la más probable; después de tres o más, h_5 (la terrorífica bolsa con todos los caramelos de lima) es la más probable. Después de 10, estamos bastante seguros de nuestro destino. La Figura 20.1(b) muestra la predicción de la probabilidad de que el siguiente caramelo sea de lima, basada en la Ecuación (20.2). Como se podía esperar, se incrementa de forma monótona hacia 1.

El ejemplo muestra que, a la larga, *la verdadera hipótesis domina la predicción bayesiana*. Esto es característico del aprendizaje Bayesiano. Para cualquier a priori fija que no excluya la hipótesis verdadera, la probabilidad a posteriori de cualquier hipótesis falsa finalmente desaparecerá, simplemente porque la probabilidad de generar datos «no característicos» de forma indefinida es cada vez más pequeña. (Esta apreciación es análoga a una realizada en la explicación del PAC-aprendizaje en el Capítulo 18.) Más importante, la predicción bayesiana es *óptima*, tanto si el conjunto de datos es pequeño,

Figura 20.1 (a) Probabilidades a posteriori $P(h_i|d_1, ..., d_N)$ calculadas a partir de la Ecuación (20.1). El número de observaciones N va de 1 a 10, y cada observación es una observación de un caramelo de lima. (b) Predicción bayesiana $P(d_{N+1} = lima|d_1, ..., d_N)$ a partir de la Ecuación (20.2).

[2] Ya mencionamos antes que las bolsas de caramelos son muy grandes; en otro caso no se cumpliría la asunción de i.i.d. Técnicamente, es más correcto (pero menos higiénico) volver a envolver cada caramelo después de inspeccionarlo y devolverlo a la bolsa.

como si es grande. Dada la hipótesis a priori, cualquier otra predicción será correcta con menos frecuencia.

Por supuesto, la optimalidad del aprendizaje Bayesiano tiene un precio. En los problemas reales de aprendizaje, el espacio de hipótesis es normalmente muy grande o infinito, como vimos en el Capítulo 18. En algunos casos, el cálculo del sumatorio de la Ecuación (20.2) (o la integración en el caso continuo) es tratable, pero en la mayoría de los casos debemos recurrir a métodos aproximados o simplificados.

MÁXIMO A POSTERIORI

Una aproximación muy común, que se adopta normalmente en la ciencia, es hacer predicciones basadas en una única hipótesis, la *más probable,* es decir, una h_i que maximice $P(h_i|\mathbf{d})$. Frecuentemente, esto se denomina **máximo a posteriori** o hipótesis MAP. Las predicciones que se realizan de acuerdo a una hipótesis MAP, h_{MAP}, son aproximadamente bayesianas, hasta el punto de que $\mathbf{P}(X|\mathbf{d}) \approx \mathbf{P}(X|h_{\text{MAP}})$. En nuestro ejemplo de los caramelos, $h_{\text{MAP}} = h_5$ después de tres caramelos de lima, por lo tanto el aprendizaje MAP predice que el cuarto caramelo será de lima con probabilidad 1,0 (una predicción mucho más peligrosa que la predicción bayesiana de 0,8 que se muestra en la Figura 20.1). A medida que llegan más datos, la predicción MAP y la bayesiana se van acercando, porque las competidoras con la hipótesis MAP se van haciendo menos y menos probables. Aunque nuestro ejemplo no muestra esto, encontrar la hipótesis MAP es normalmente más sencillo que el aprendizaje Bayesiano, porque requiere resolver un problema de optimización en vez de un gran sumatorio (o integración). Veremos ejemplos de esto último en este capítulo.

Tanto en el aprendizaje Bayesiano como en el aprendizaje MAP, la hipótesis a priori $P(h_i)$ tiene un papel importante. En el Capítulo 18, vimos que puede ocurrir **sobreajuste** cuando el espacio de hipótesis es muy expresivo, y por lo tanto, contiene muchas hipótesis que se ajustan bien al conjunto de datos. Más que imponer un límite arbitrario al espacio de hipótesis, los métodos de aprendizaje Bayesiano y MAP utilizan la distribución a priori para *penalizar la complejidad*. Típicamente, las hipótesis más complejas tienen una probabilidad a priori más pequeña, en parte porque normalmente hay muchas más hipótesis complejas que simples. Por otro lado, las hipótesis más complejas tienen más capacidad para ajustarse a los datos. (En el caso extremo, una tabla de búsqueda puede reproducir exactamente los datos con probabilidad 1.) Por lo tanto, la hipótesis a priori expresa un compromiso entre la complejidad de una hipótesis y su grado de ajuste con los datos.

Podemos observar el efecto de este compromiso más claramente en el caso lógico, donde H contiene sólo hipótesis *determinísticas*. En este caso, $P(\mathbf{d}|h_i)$ es 1 si h_i es consistente y 0 en otro caso. Observando la Ecuación (20.1), se puede ver que h_{MAP} será la *teoría lógica más simple consistente con los datos*. Por lo tanto, el aprendizaje máximo *a posteriori* incluye de forma natural la navaja de Ockham.

Otra idea, relativa al compromiso entre complejidad y grado de ajuste proviene del algoritmo de la Ecuación (20.1). Elegir h_{MAP} para maximizar $P(\mathbf{d}|\, h_i)\, P(h_i)$ es equivalente a minimizar

$$-\log_2 P(\mathbf{d}|h_i) - \log_2 P(h_i)$$

Utilizando la conexión entre la codificación de la información y la probabilidad, que se introdujo en el Capítulo 18, se puede ver que el término $-\log_2 P(h_i)$ es igual al número

de bits que se requieren para especificar h_i. Además, $-\log_2 P(\mathbf{d}|h_i)$, es el número de bits adicionales que se requieren para especificar los datos, dada la hipótesis. (Para ver esto considere que no se necesitan bits si la hipótesis predice exactamente los datos —como con h_5 y la tira de caramelos de lima— y $\log_2 1 = 0$). Así, el aprendizaje MAP elige la hipótesis que proporciona una *compresión* máxima de los datos. La misma tarea se resuelve directamente por el método de aprendizaje de **mínima longitud de descripción**, o MDL, que en lugar de trabajar con probabilidades, intenta minimizar el tamaño de las codificaciones de las hipótesis y de los datos.

MÍNIMA LONGITUD DE DESCRIPCIÓN

Se obtiene una simplificación más si se asume una distribución **uniforme** a priori sobre el espacio de hipótesis. En este caso, el aprendizaje MAP se reduce a elegir una h_i que maximice $P(\mathrm{d}|H_i)$. Esta hipótesis se denomina hipótesis de **máxima verosimilitud** (ML), h_{ML}. El aprendizaje de máxima verosimilitud es muy común en estadística, una disciplina en la que muchos investigadores desconfían de la naturaleza subjetiva de las hipótesis a priori. Es un enfoque razonable cuando *a priori* no hay razón para preferir una hipótesis sobre otra, por ejemplo, cuando todas las hipótesis son igual de complejas. Proporciona una buena aproximación al aprendizaje Bayesiano y MAP cuando el conjunto de datos es muy grande, porque los datos imperan sobre la distribución a priori de la hipótesis, pero presenta problemas (como veremos) con conjuntos de datos pequeños.

MÁXIMA VEROSIMILITUD

20.2 Aprendizaje con datos completos

APRENDIZAJE DE PARÁMETROS

DATOS COMPLETOS

Nuestro desarrollo de métodos de aprendizaje estadísticos comienza con la tarea más sencilla: **aprendizaje de parámetros** con **datos completos**. Una tarea de aprendizaje de parámetros supone encontrar los parámetros numéricos para un modelo probabilístico cuya estructura está fijada. Por ejemplo, podríamos estar interesados en aprender las probabilidades condicionales de una red bayesiana con una estructura dada. Los datos son completos cuando incluyen valores para cada variable del modelo de probabilidad que está siendo aprendido. Los datos completos simplifican en gran medida el problema de aprendizaje de parámetros de un modelo complejo. También veremos brevemente el problema de aprendizaje de la estructura.

Aprendizaje del parámetro de máxima verosimilitud: modelos discretos

Suponga que compramos una bolsa de caramelos de lima y cereza en una nueva fábrica cuyas proporciones de lima y cereza son completamente desconocidas, es decir, la fracción puede ser cualquier valor entre 0 y 1. En este caso, tenemos una hipótesis continua. El **parámetro** en este caso, que llamaremos θ, es la proporción de caramelos de cereza, y la hipótesis es h_θ. (La proporción de caramelos de lima es exactamente 1-θ.) Si asumimos que *a priori* todas las proporciones son igual de probables, el enfoque de máxima verosimilitud es razonable. Si modelizamos la situación con una red bayesiana, necesitamos exactamente una variable aleatoria, *Sabor* (el sabor de un caramelo ele-

gido aleatoriamente de la bolsa), que tiene valores *cereza* y *lima*, donde la probabilidad de *cereza* es θ (*véase* Figura 20.2(a)). Ahora suponga que desenvolvemos N caramelos, de los cuales c son de cereza y $\ell = N - c$ son de lima. De acuerdo con la ecuación (20.3), la verosimilitud de este conjunto de datos particular es

LOGARITMO DE LA VEROSIMILITUD

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c \cdot (1-\theta)^\ell$$

La hipótesis de máxima verosimilitud viene dada por el valor θ que maximiza esta expresión. El mismo valor se obtiene maximizando el **logaritmo de la verosimilitud**

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = \sum_{j=1}^{N} \log P(d_j|h_\theta) = c \log \theta + \ell \log(1-\theta)$$

(Tomando logaritmos, reducimos el producto a una suma sobre los datos, que normalmente es más fácil de maximizar.) Para encontrar el valor de máxima verosimilitud de θ, derivamos L respecto a θ e igualamos a cero la expresión que resulta:

$$\frac{dL(\mathbf{d}|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell} = \frac{c}{N}$$

Por lo tanto, la hipótesis de máxima verosimilitud h_{ML}, ¡afirma que la proporción actual de caramelos de cereza en la bolsa es igual a la proporción observada de caramelos desenvueltos!

Parece que hemos hecho mucho trabajo para descubrir lo obvio. Aunque de hecho, hemos aplicado un método estándar de aprendizaje de parámetros de máxima verosimilitud:

1. Expresar la verosimilitud de los datos como una función de los parámetros.
2. Derivar el logaritmo de la verosimilitud respecto a cada parámetro.
3. Encontrar los valores de los parámetros que hacen las derivadas iguales a cero.

Figura 20.2 (a) Modelo de red bayesiana para el caso de caramelos con una proporción desconocida de cerezas y limas. (b) Modelo para el caso en el que el color del envoltorio depende (de forma probabilística) del sabor del caramelo.

Normalmente, el paso más complicado es el último. En nuestro ejemplo era trivial, pero veremos que en muchos casos necesitamos recurrir a algoritmos iterativos u otras técnicas de optimización numérica, como se describía en el Capítulo 14. El ejemplo también ilustra un problema significativo que en general se presenta con el aprendizaje de máxima verosimilitud: *cuando el conjunto de datos es tan pequeño que algunos eventos no se han observado (por ejemplo, sin caramelos de cereza) la hipótesis de máxima verosimilitud asigna probabilidad cero a esos eventos*. Para evitar este problema se han utilizado varios trucos, como inicializar los recuentos de cada evento a 1 en lugar de a cero.

Veamos otro ejemplo. Suponga que este nuevo fabricante de caramelos quiere dar una pista al consumidor y utiliza caramelos cuyos envoltorios son rojos y verdes. El *Envoltorio* para cada caramelo se selecciona de forma *probabilística*, de acuerdo a alguna distribución condicional desconocida, dependiendo del sabor. El modelo de probabilidad correspondiente se muestra en la Figura 20.2(b). Observe que tiene tres parámetros: θ, θ_1, y θ_2. Con estos parámetros, la verosimilitud de que haya un caramelo de cereza en un envoltorio verde se puede obtener a partir de la semántica estándar de las redes bayesianas (Apartado 14.2):

$$\begin{aligned}
&P(Sabor = cereza, Envoltorio = verde | h_{\theta,\theta_1,\theta_2}) \\
&\quad = P(Sabor = cereza | h_{\theta,\theta_1,\theta_2}) P(Envoltorio = verde | Sabor = cereza, h_{\theta,\theta_1,\theta_2}) \\
&\quad = \theta \cdot (1 - \theta_1)
\end{aligned}$$

Ahora, desenvolvemos N caramelos, de los cuales c son de cereza y l de lima. Los recuentos de envoltorios son los siguientes: r_c de los caramelos de cereza tienen envoltorio rojo y g_c tienen envoltorio verde, mientras que r_l de los de lima tienen envoltorio rojo y g_l lo tienen verde. La verosimilitud de los datos viene dada por:

$$P(\mathbf{d} | h_{\theta,\theta_1,\theta_2}) = \theta^c (1 - \theta)^\ell \cdot \theta_1^{r_c} (1 - \theta_1)^{g_c} \cdot \theta_2^{r_\ell} (1 - \theta_2)^{g_\ell}$$

Esto parece bastante horrible, pero tomar logaritmos ayuda:

$$L = [c \log\theta + \ell \log(1-\theta)] + [r_c \log\theta_1 + g_c \log(1-\theta_1)] + [r_\ell \log\theta_2 + g_\ell \log(1-\theta_2)]$$

El beneficio que se obtiene tomando logaritmos es claro: el logaritmo de la verosimilitud es la suma de los tres términos, cada uno de los cuales contiene un parámetro simple. Si hacemos las derivadas respecto a cada parámetro y las fijamos a cero, obtendremos tres ecuaciones independientes, cada una de las cuales contiene exactamente un parámetro:

$$\begin{aligned}
\frac{\partial L}{\partial \theta} &= \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+l} \\
\frac{\partial L}{\partial \theta_1} &= \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \quad \Rightarrow \quad \theta_1 = \frac{r_c}{r_c + g_c} \\
\frac{\partial L}{\partial \theta_2} &= \frac{r_l}{\theta_2} - \frac{g_l}{1-\theta_2} = 0 \quad \Rightarrow \quad \theta_2 = \frac{r_l}{r_l + g_l}
\end{aligned}$$

La solución para θ es la misma de antes. La solución para θ_1, la probabilidad de que un caramelo de cereza tenga un envoltorio rojo, es la fracción observada de caramelos de cereza con envoltorios rojos, y para θ_2 de forma similar.

Estos resultados son muy reconfortantes, y es muy fácil ver que pueden ser extendidos a cualquier red bayesiana cuyas probabilidades condicionales vengan representadas por tablas. El punto más importante es que, *con datos completos, el problema de aprendizaje de parámetros por máxima verosimilitud para una red bayesiana se descompone en problemas de aprendizaje separados para cada uno de los parámetros*[3]. El segundo punto es que los valores de los parámetros para una variable, dados su padres, son exactamente las frecuencias observadas de los valores de la variable para cada combinación de valores de los padres. Como antes, debemos tener cuidado para evitar ceros cuando el conjunto de datos es pequeño.

Modelos de Bayes simples (Naive Bayes)

Probablemente el modelo de red bayesiana que más comúnmente se ha utilizado en aprendizaje automático es el modelo de **Bayes simple**. En este modelo, la variable «clase» C (que se va a predecir) es la raíz y las variables «atributo» X_i son las hojas. El modelo es simple o «ingenuo» porque asume que los atributos son condicionalmente independientes entre sí, dada la clase. (El modelo de la Figura 20.2(b) es un modelo de Bayes ingenuo con exactamente un atributo.) Asumiendo variables Booleanas, los parámetros son

$$\theta = P(C = verdadero),\ \theta_{i1} = P(X_i = verdadero | C = verdadero),\ \theta_{i2} = \\ = P(X_i = verdadero | C = falso)$$

Los valores de los parámetros de máxima verosimilitud se encuentran exactamente de la misma forma que para la Figura 20.2(b). Una vez que el modelo ha sido entrenado de esta forma, puede ser utilizado para clasificar nuevos ejemplos para los cuales la variable clase C no se ha observado. Con los valores de atributo observados $x_1, ..., x_n$, la probabilidad para cada clase viene dada por

$$\mathbf{P}(C|x_1,...,x_n) = \alpha\, \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C)$$

Se puede obtener una predicción determinística, eligiendo la clase más probable. La Figura 20.3 muestra la curva de aprendizaje para este método cuando se aplica al problema del restaurante del Capítulo 18. El método aprende bastante bien, pero no tan bien como el aprendizaje con árboles de decisión; presumiblemente, esto ocurre porque la hipótesis verdadera (que es un árbol de decisión) no se puede representar exactamente utilizando un modelo de Bayes simple. El aprendizaje de Bayes simple funciona sorprendentemente bien en una gran variedad de aplicaciones; la versión potenciada (*boosted*) (Ejercicio 20.5) es uno de los algoritmos de aprendizaje de propósito general más efectivos. Funciona bien con problemas muy grandes: con n atributos Booleanos, hay exactamente $2n + 1$ parámetros, y *no se requiere búsqueda para encontrar h_{ML}, la hipótesis de máxima verosimilitud de Bayes simple*. Finalmente, el aprendizaje de Bayes simple no presenta dificultades con datos con ruido y puede proporcionar predicciones probabilísticas cuando sea apropiado.

[3] *Véase* el Ejercicio 20.7 para el caso no tabulado, donde cada parámetro afecta a varias probabilidades condicionales.

Figura 20.3 La curva de aprendizaje para el aprendizaje de Bayes simple aplicado al problema del restaurante del Capítulo 18; la curva de aprendizaje para el aprendizaje con árboles de decisión se muestra para hacer la comparación.

Aprendizaje de parámetros de máxima verosimilitud: modelos continuos

Los modelos probabilísticos continuos, como el modelo **lineal-Gaussiano**, fueron introducidos en el Apartado 14.3. Es importante saber cómo se aprenden modelos continuos a partir de los datos, ya que las variables continuas siempre están presentes en las aplicaciones reales. Los principios para el aprendizaje de máxima verosimilitud son idénticos a los del caso discreto.

Empecemos con un caso sencillo: el aprendizaje de parámetros de una función de densidad gaussiana sobre una variable simple. Esto es, los datos son generados como sigue:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Los parámetros de este modelo son la media μ y la desviación estándar σ. (Observe que la «constante» de normalización depende de σ, por lo que la podemos ignorar.) Sean $x_1, ..., x_N$ los valores observados. El logaritmo de la verosimilitud es

$$L = \sum_{j=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2}$$

Fijando las derivadas a cero, como es usual, obtenemos

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2} \Sigma_{j=1}^{N} (x_j - \mu) = 0 \quad &\Rightarrow \quad \mu = \frac{\Sigma_j x_j}{N} \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3} \Sigma_{j=1}^{N} (x_j - \mu)^2 = 0 \quad &\Rightarrow \quad \sigma = \sqrt{\frac{\Sigma_j (x_j - \mu)^2}{N}} \end{aligned} \tag{20.4}$$

Es decir, el valor de máxima verosimilitud de la media es la media de la muestra y el valor de máxima verosimilitud de la desviación estándar es la raíz cuadrada de la varianza de la muestra. Otra vez, estos son resultados reconfortantes que confirman la práctica del «sentido común».

Ahora considere un modelo Gaussiano lineal con un padre continuo X y un hijo continuo Y. Como se explicó en el Apartado 14.3, Y tiene una distribución gaussiana cuya media depende linealmente del valor de X, y cuya desviación estándar es fija. Para aprender la distribución condicional $P(Y|X)$, podemos maximizar la verosimilitud condicional

$$P(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}} \tag{20.5}$$

Aquí, los parámetros son θ_1, θ_2 y σ. Los datos son una colección de pares (x_j, y_j), como se ilustra en la Figura 20.4. Utilizando los métodos habituales (Ejercicio 20.6), podemos encontrar los valores de máxima verosimilitud de los parámetros. Aquí, hay que hacer una distinción. Si consideramos exactamente los parámetros θ_1, θ_2 que definen las relaciones lineales entre x e y, está claro que maximizar el logaritmo de la verosimilitud respecto a estos parámetros es lo mismo que minimizar el numerador en el exponente de la Ecuación (20.5):

$$E = \sum_{j=1}^{N} (y_j - (\theta_1 x_j + \theta_2))^2$$

ERROR

SUMA DE ERRORES CUADRADOS

REGRESIÓN LINEAL

La cantidad $(y_j - (\theta_1 x_j + \theta_2))$ es el **error** para (x_j, y_j) (que es la diferencia entre el valor actual y_j y el valor predicho $(\theta_1 x_j + \theta_2)$) por lo tanto, E es la tan conocida **suma de los errores cuadrados**. Esta es la cantidad que se minimiza por el procedimiento de **regresión lineal** estándar. Ahora podemos entender por qué: minimizar la suma de los errores cuadrados proporciona el modelo de línea recta de máxima verosimilitud, *cuando los datos han sido generados con ruido Gaussiano de varianza fija*.

Figura 20.4 (a) Un modelo lineal Gaussiano descrito como $y = \theta_1 x + \theta_2$ más el ruido Gaussiano con varianza fija. (b) Un conjunto de 50 puntos generados a partir de este modelo.

Aprendizaje de parámetros Bayesiano

El aprendizaje de máxima verosimilitud da lugar a procedimientos muy simples, pero presenta deficiencias serias con conjuntos de datos pequeños. Por ejemplo, después de ver un caramelo de cereza, la hipótesis de máxima verosimilitud es que la bolsa es de 100 por cien de caramelos de cereza (es decir, $\theta = 1{,}0$). A menos que una hipótesis a priori sea que las bolsas deben contener todos los caramelos de cereza o todos de lima, ésta no es una conclusión razonable. El enfoque Bayesiano para el aprendizaje de parámetros sitúa una hipótesis a priori sobre los valores posibles de los parámetros y actualiza esta distribución a medida que los datos van llegando.

El ejemplo de los caramelos de la Figura 20.2(a) tiene un parámetro θ: la probabilidad de que un caramelo seleccionado aleatoriamente sea de sabor cereza. Desde el punto de vista Bayesiano, θ es el valor (desconocido) de una variable aleatoria Θ; la hipótesis a priori es exactamente la distribución a priori $\mathbf{P}(\Theta)$. Así, $P(\Theta = \theta)$ es la probabilidad de que la bolsa tenga una fracción θ de caramelos de cereza.

Si el parámetro θ puede ser cualquier valor entre 0 y 1, $\mathbf{P}(\Theta)$ debe ser una distribución continua que es distinta de cero sólo entre 0 y 1, y cuya integral es 1. La densidad uniforme $P(\theta) = \mathrm{U}[0,1](\theta)$ es una candidata. (*Véase* Capítulo 13). Resulta que la densidad uniforme es un miembro de la familia de **distribuciones beta**. Cada distribución beta se define mediante dos **hiperparámetros**[4] a y b tal que

DISTRIBUCIONES BETA

HIPERPARÁMETROS

$$\mathrm{beta}[a, b](\theta) = \alpha\theta^{a-1}(1-\theta)^{b-1} \tag{20.6}$$

para θ en el rango [0,1]. La constante de normalización α depende de a y b. (*Véase* Ejercicio 20.8). La Figura 20.5 muestra qué aspecto tiene la distribución para varios valores de a y b. La media de la distribución es $a / (a + b)$, por lo tanto, valores grandes de a sugieren la creencia de que Θ está más cercana a 1 que a 0. Valores grandes de $a + b$ ha-

Figura 20.5 Ejemplos de la distribución beta[a, b] para valores diferentes de [a, b].

[4] Se llaman hiperparámetros porque parametrizan una distribución sobre θ, que es por sí misma un parámetro.

cen la distribución más puntiaguda, insinuando mayor certeza sobre el valor de Θ. Así, la familia beta proporciona un rango útil de posibilidades para la hipótesis a priori.

Además de su flexibilidad, la familia beta tiene otra propiedad maravillosa: si Θ sigue a priori una beta[a,b], después de que un punto de los datos sea observado, la distribución a posteriori para Θ es también una distribución beta. Se dice que la familia beta es la **conjugada a priori** para la familia de distribuciones para una variable booleana[5]. Veamos cómo funciona esto. Suponga que observamos un caramelo de cereza; entonces

CONJUGADA A PRIORI

$$\begin{aligned} P(\theta|D_1 = cereza) &= \alpha P(D_1 = cereza|\theta)P(\theta) \\ &= \alpha'\theta \cdot \text{beta}[a, b](\theta) = \alpha'\theta \cdot \theta^{a-1}(1-\theta)^{b-1} \\ &= \alpha'\theta^{a}(1-\theta)^{b-1} = \text{beta}[a+1, b](\theta) \end{aligned}$$

Así, después de observar un caramelo de cereza, simplemente incrementamos el parámetro a para obtener la distribución a posteriori; de forma similar, después de observar un caramelo de lima, incrementamos el parámetro b. De esta forma, podemos ver los hiperparámetros a y b como **contadores virtuales**, en el sentido de que una beta[a,b] *a priori*, viendo $a - 1$ como los caramelos de cereza actuales y $b - 1$ como los caramelos de lima actuales, se comporta exactamente como si hubiéramos empezado con una uniforme a priori, beta[1, 1].

CONTADORES VIRTUALES

Examinando la secuencia de distribuciones beta para incrementar los valores de a y b, manteniendo las proporciones fijas, podemos ver cómo la distribución a posteriori sobre el parámetro Θ varía a medida que llegan los datos. Por ejemplo, suponga que la bolsa de caramelos actual es del tipo 75 por ciento de cereza. La Figura 20.5(b) muestra la secuencia beta[3,1], beta[6,2], beta[30,10]. Claramente, la distribución converge a un pico estrecho alrededor del valor verdadero de Θ. Para grandes conjuntos de datos, el aprendizaje Bayesiano (al menos en este caso) tiende a proporcionar los mismos resultados que el aprendizaje de máxima verosimilitud.

La red de la Figura 20.2(b) tiene tres parámetros θ, θ_1 y θ_2, donde θ_1 es la probabilidad de que un envoltorio rojo cubra un caramelo de cereza y θ_2 es la probabilidad de que un envoltorio rojo cubra un caramelo de lima. La hipótesis bayesiana *a priori* debe cubrir los tres parámetros, es decir, necesitamos especificar $\mathbf{P}(\Theta, \Theta_1, \Theta_2)$. Normalmente asumiremos **independencia de parámetros**:

INDEPENDENCIA DE PARÁMETROS

$$\mathbf{P}(\Theta, \Theta_1, \Theta_2) = \mathbf{P}(\Theta)\mathbf{P}(\Theta_1)\mathbf{P}(\Theta_2)$$

Con esta suposición, cada parámetro puede tener su propia distribución beta que se actualiza de forma separada a medida que llegan los datos.

Una vez que tenemos la idea de que los parámetros desconocidos se pueden representar con variables aleatorias como Θ, es natural incorporarlos en la propia red bayesiana. Para hacer esto, necesitamos también hacer copias de las variables que describen cada instancia. Por ejemplo, si hemos observado tres caramelos, necesitamos $Sabor_1$, $Sabor_2$, $Sabor_3$, y $Envoltorio_1$, $Envoltorio_2$, $Envoltorio_3$. El parámetro variable Θ determina la probabilidad de cada variable $Sabor_i$:

$$\mathbf{P}(Sabor_i = cereza|\Theta = \theta) = \theta$$

[5] Otras conjugadas a priori son la familia **Dirichlet** para los parámetros de una distribución multivaluada y la familia **Normal-Wishart** para los parámetros de una distribución gaussiana. *Véase* Bernardo y Smith (1994).

De forma similar, las probabilidades de los envoltorios dependen de Θ_1 y Θ_2, por ejemplo,

$$\mathbf{P}(Envoltorio_i = rojo | Sabor_i = \Theta_1 = \theta_1) = \theta_1$$

Ahora, el proceso de aprendizaje bayesiano completo se puede formular como un problema de *inferencia* en una red de Bayes construida de forma adecuada, como se muestra en la Figura 20.6. La predicción de una nueva instancia se realiza simplemente añadiendo nuevas variables instancia a la red, algunas de las cuales se consultan. Esta formulación del aprendizaje y la predicción deja claro que el aprendizaje Bayesiano no requiere «principios de aprendizaje» extra. Además, *hay, en esencia, sólo un algoritmo de aprendizaje*, el algoritmo de inferencia de redes bayesianas.

Aprendizaje de la estructura de las redes bayesianas

Hasta ahora hemos asumido que la estructura de la red de Bayes viene dada y hemos tratado de aprender los parámetros. La estructura de la red representa conocimiento causal básico sobre el dominio, que frecuentemente resulta sencillo para un experto, o incluso para un simple usuario. Sin embargo, en algunos casos el modelo causal puede no estar disponible o ser objeto de controversias (por ejemplo, algunas sociedades han reivindicado en muchas ocasiones que fumar no causa cáncer) por lo tanto, es importante entender cómo se puede aprender una estructura de Bayes a partir de los datos. Hoy en día, los algoritmos de aprendizaje de la estructura están todavía en su infancia, por lo que mostraremos solamente un breve esquema de las ideas principales.

Figura 20.6 Una red bayesiana que corresponde a un proceso de aprendizaje. Las distribuciones a posteriori para las variables parámetro Θ, Θ_1, y Θ_2 se pueden inferir a partir de las distribuciones a priori y la evidencia en las variables $Sabor_i$ y $Envoltorio_i$.

El enfoque más obvio es *buscar* un modelo bueno. Podemos empezar con un modelo que no contiene arcos e ir añadiendo padres a cada nodo, fijando los parámetros con los métodos que ya hemos mencionado antes y midiendo la precisión del modelo resultante. Alternativamente, podemos empezar con la estructura inicial que nos indique la intuición y utilizar búsqueda en escalada (*hill-climbing*) o recocido simulado (*simulated annealing*) para hacer modificaciones, refinando los parámetros después de cada cambio en la estructura. Las modificaciones pueden incluir invertir, añadir o borrar arcos. No debemos introducir ciclos en el proceso, por lo que muchos algoritmos suponen que viene dado un orden para las variables, y que los padres de un nodo sólo pueden ser aquellos que aparecen antes que él según este orden (exactamente como en el proceso de construcción del Capítulo 14). Para una generalidad completa, también necesitamos buscar sobre todos los órdenes posibles.

Existen dos métodos alternativos para decidir cuándo se ha encontrado una buena estructura. El primero es comprobar si las afirmaciones de independencia condicional implícitas en la estructura se satisfacen realmente por los datos. Por ejemplo, el uso de un modelo de Bayes simple para el problema del restaurante asume que

$$\mathbf{P}(Vier/Sab, Bar|Esperar) = \mathbf{P}(Vier/Sab|Esperar)\mathbf{P}(Bar|Esperar)$$

y podemos comprobar si la misma ecuación se verifica entre las correspondientes frecuencias condicionales de los datos. Pero, incluso si la estructura describe la verdadera naturaleza causal del dominio, las fluctuaciones estadísticas en el conjunto de datos hace que la ecuación nunca se satisfaga de forma *exacta*, por lo tanto, necesitamos desarrollar un test estadístico adecuado para comprobar si hay suficiente evidencia de que la hipótesis de independencia se ha violado. La complejidad de la red resultante dependerá del umbral que se utilice en este test (un test de independencia es más estricto y el peligro de sobreajuste es mayor cuantos más arcos añade).

Un enfoque más consistente con las ideas de este capítulo consiste en medir en qué grado el modelo propuesto explica los datos (en un sentido probabilístico). Sin embargo, debemos tener cuidado al hacer esto. Si intentamos encontrar exactamente la hipótesis de máxima verosimilitud, acabaremos con una red totalmente conectada porque añadir más padres a un nodo no decrementa la verosimilitud (Ejercicio 20.9). Estamos obligados a penalizar de alguna forma la complejidad del modelo. El enfoque MAP (o MDL) simplemente resta una penalización de la verosimilitud de cada estructura (después del ajuste de parámetros) antes de comparar diferentes estructuras. El enfoque Bayesiano sitúa una distribución conjunta a priori sobre las estructuras y parámetros. Normalmente hay muchas estructuras que considerar (superexponencial en el número de variables), por eso la mayoría de los usuarios utilizan MCMC para muestrear sobre las estructuras.

Penalizar la complejidad (mediante métodos MAP o bayesianos) introduce una conexión importante entre la estructura óptima y la naturaleza de la representación de las distribuciones condicionales en la red. Con distribuciones tabulares, la penalización de la complejidad para la distribución de un nodo crece exponencialmente con el número de padres, pero con distribuciones de interacción disyuntiva (*noisy-OR*) crece linealmente. Esto significa que con modelos *noisy-OR* (u otros modelos parametrizados de forma compacta), las estructuras aprendidas tienden a tener más padres que con distribuciones tabulares.

20.3 Aprendizaje con variables ocultas: el algoritmo EM

VARIABLES LATENTES

En la sección anterior tratamos el caso completamente observable. Muchos problemas reales tienen **variables ocultas** (a veces denominadas **variables latentes**) que no son observables en los datos disponibles para el aprendizaje. Por ejemplo, los registros médicos muchas veces incluyen los síntomas observados, el tratamiento aplicado, y quizás el resultado del tratamiento, ¡pero no contienen una observación directa de la propia enfermedad![6] Se podría preguntar: «Si la enfermedad no se ha observado, ¿por qué no construir un modelo sin ella?». La respuesta está en la Figura 20.7, que muestra un modelo de diagnóstico pequeño y ficticio para una enfermedad de corazón. Hay tres factores disponibles que se pueden observar y tres síntomas observables (que es muy deprimente nombrar). Se asume que cada variable tiene tres valores posibles (por ejemplo, *nada*, *moderado* y *severo*). Eliminando la variable oculta de la red en (a) se obtiene la red en (b); el número total de parámetros aumenta de 78 a 708. Así, *las variables latentes pueden reducir dramáticamente el número de parámetros que se requieren para especificar una red bayesiana*. En realidad, esto puede reducir dramáticamente la cantidad de datos que se necesitan para aprender los parámetros.

Las variables ocultas son importantes, pero complican el problema de aprendizaje. En la Figura 20.7(a), por ejemplo, no es obvio cómo aprender la distribución condicional para *EnfermedadCorazón*, dados sus padres, porque no conocemos el valor de *EnfermedadCorazón* en cada caso; este mismo problema surge en el aprendizaje

Figura 20.7 (a) Una red sencilla de diagnóstico de una enfermedad del corazón, que tiene una variable oculta. Cada variable tiene tres valores posibles y es etiquetada con el números de parámetros independientes de su distribución condicional; el número total es de 78. (b) La red equivalente con *EnfermedadCorazón* eliminada. Observe que las variables síntoma no son condicionalmente independientes dados sus padres. Esta red requiere 708 parámetros.

[6] Algunos registros contienen el diagnóstico sugerido por el médico, pero esto es una consecuencia causal de los síntomas, que en realidad son causados por la enfermedad.

MAXIMIZACIÓN DE LA ESPECTATIVA

de las distribuciones de los síntomas. Esta sección describe un algoritmo de **maximización de la expectativa** o EM, que resuelve este problema de una forma muy general. Mostraremos tres ejemplos y proporcionaremos una descripción general. El algoritmo parece mágico al principio, pero una vez que se ha desarrollado la intuición, uno puede encontrar aplicaciones para EM en una gran variedad de problemas de aprendizaje.

Agrupamiento no supervisado: aprendizaje de mezclas de gaussianas

AGRUPAMIENTO NO SUPERVISADO

El **agrupamiento no supervisado** es el problema de discernir múltiples categorías en una colección de objetos. El problema es no supervisado porque no se proporcionan las etiquetas de las categorías. Por ejemplo, suponga que registramos el espectro de cientos de miles de estrellas; el espectro revela que hay diferentes *tipos* de estrellas, si es así ¿cuántas y cuáles son sus características? Todos estamos familiarizados con términos como «gigante roja» y «enana blanca», pero las estrellas no llevan estas etiquetas escritas en sus sombreros; los astrónomos hubieron de realizar agrupamientos no supervisados para identificar estas categorías. Otros ejemplos incluyen la identificación de especies, género, órdenes, etc., en la taxonomía de Linnean de los organismos y la creación de clases naturales para categorizar objetos ordinarios (*véase* Capítulo 10).

El agrupamiento no supervisado comienza con los datos. La Figura 20.8(a) muestra unos datos que contienen 500 puntos, cada uno de los cuales especifica los valores de dos atributos continuos. Los puntos pueden corresponder a estrellas y los atributos pueden corresponder a intensidades espectrales de dos frecuencias particulares. Necesitamos entender qué tipo de distribución de probabilidad pueden haber generado los datos. El agrupamiento presupone que los datos son generados a partir de una **distribución mezcla**, P. Cada distribución tiene k **componentes**, cada una de las cuales es propiamente una distribución. Un punto de los datos se genera eligiendo primero una componente y

DISTRIBUCIÓN MEZCLA

COMPONENTE

Figura 20.8 (a) 500 puntos de los datos en dos dimensiones, que sugieren la presencia de tres grupos. (b) Un modelo de mezclas de gaussianas con tres componentes; los pesos (de izquierda a derecha) son 0,2, 0,3 y 0,5. Los datos de (a) fueron generados a partir de este modelo. (c) El modelo reconstruido por EM a partir de los datos de (b).

después generando una muestra a partir de esa componente. Sea la variable aleatoria C que denota la componente, con valores 1, ..., k; la distribución mixtura viene dada por

$$P(\mathbf{x}) = \sum_{i=1}^{k} P(C = i)P(\mathbf{x}|C = i)$$

donde $\mathbf{x}$ se refiere a los valores de los atributos para los puntos de los datos. Para datos continuos, una elección natural de las distribuciones componentes es la gaussiana multivariante, que da lugar a la denominada familia de distribuciones de **mezclas de gaussianas**. Los parámetros de una mezcla de gaussianas son $w_i = P(C = i)$ (el peso de cada componente), μ_i (la media de cada componente), y Σ_i (la covarianza de cada componente). La Figura 20.8(b) muestra una mixtura de tres gaussianas; esta mezcla es, de hecho, la fuente de los datos de (a).

MEZCLAS GAUSSIANAS

Entonces, el problema de agrupamiento no supervisado es recuperar un modelo de mixtura como el que aparece en la Figura 20.8(b) a partir de datos crudos como los de la Figura 20.8(a). Claramente, si *conociéramos* qué componente genera cada punto de los datos sería fácil recuperar las componentes gaussianas: podríamos seleccionar exactamente todos los puntos de los datos a partir de una componente dada y después aplicar (una versión multivariante) de la Ecuación (20.4) para ajustar los parámetros de una gaussiana a un conjunto de datos. Por otro lado, si *conociéramos* los parámetros de cada componente podríamos, al menos en un sentido probabilístico, asignar cada punto de los datos a una componente. El problema es que no conocemos las asignaciones ni los parámetros.

La idea básica de EM en este contexto es *fingir* que conocemos los parámetros del modelo e inferir la probabilidad de que cada punto de los datos pertenezca a cada componente. Después de esto reajustamos las componentes a los datos, donde cada componente se ajusta al conjunto de datos completo, donde cada punto tiene un peso igual a la probabilidad de que pertenezca a esa componente. El proceso se itera hasta que converja. Esencialmente, estamos «completando» los datos, infiriendo las distribuciones de probabilidad de las variables ocultas (a qué componente pertenece cada punto de los datos) basándonos en el modelo actual. Para la mezcla de gaussianas, inicializamos los parámetros del modelo de mezcla aleatoriamente y después iteramos los dos pasos siguientes:

1. Paso-E: se calculan las probabilidades $p_{ij} = P(C = i|\mathbf{x}_j)$, probabilidad de que el dato $\mathbf{x}_j$ sea generado por la componente i. Mediante la regla de Bayes, tenemos $p_{ij} = \alpha\, P(\mathbf{x}_j|C = i)\, P(C = i)$. El término $P(\mathbf{x}_j|C = i)$ es exactamente la probabilidad para $\mathbf{x}_j$ de la i-ésima gaussiana, y el término $P(C = i)$ es exactamente el parámetro peso de la i-ésima gaussiana. Definir $p_i = \Sigma_j\, p_{ij}$.
2. Paso-M: calcular la nueva media, covarianza y pesos de las componentes como sigue:

$$\mu_i \leftarrow \sum_j p_{ij}\mathbf{x}_j/p_i$$
$$\Sigma_i \leftarrow \sum_j p_{ij}\mathbf{x}_j\mathbf{x}_j^T/p_i$$
$$\omega_i \leftarrow p_i$$

VARIABLE INDICADOR

El paso-E o paso del *cálculo de esperanzas*, se puede ver como el cálculo de los valores esperados p_{ij} de las **variables indicador** ocultas Z_{ij}, donde Z_{ij} es 1 si el dato $\mathbf{x}_j$ fue generado por la i-ésima componente y 0 en otro caso. El paso-M, o paso de *maximización*, encuentra los nuevos valores de los parámetros que maximizan el logaritmo de la verosimilitud de los datos, dados los valores esperados de las variables indicador ocultas.

En la Figura 20.8(c) se muestra el modelo final que aprende EM cuando se aplica a los datos de la Figura 20.8(a); prácticamente es indistinguible del modelo original a partir del que los datos fueron generados. La Figura 20.9(a) muestra el logaritmo de la verosimilitud de los datos de acuerdo al modelo actual a medida que EM progresa. Hay dos puntos a resaltar. Primero, el logaritmo de la verosimilitud para el modelo aprendido final supera ligeramente el del modelo original a partir del que los datos fueron generados. Esto puede parecer sorprendente, pero simplemente refleja el hecho de que los datos fueron generados aleatoriamente y puede que no proporcionen un reflejo exacto del modelo subyacente. El segundo punto es que *EM incrementa el logaritmo de la verosimilitud de los datos en cada iteración*. Este hecho se puede demostrar en general. Además, bajo ciertas condiciones, se puede probar que EM alcanza un máximo local de verosimilitud. (En casos raros, podría alcanzar un punto de silla o incluso un mínimo local.) En este sentido, EM se parece al algoritmo de escalada basado en el gradiente, pero ¡observe que no tiene el parámetro de «tamaño del paso»!

Las cosas no siempre funcionan tan bien como sugiere la Figura 20.9(a). Por ejemplo, puede ocurrir que una componente gaussiana se encoja y cubra únicamente un solo punto de los datos. ¡En este caso su varianza será cero y su verosimilitud será infinito! Otro problema es que dos componentes se pueden «fundir», adquiriendo medias y varianzas idénticas y compartiendo sus puntos de los datos. Estos tipos de máximos locales degenerados constituyen problemas serios, especialmente en grandes dimensiones.

Figura 20.9 Los gráficos muestran el logaritmo de la verosimilitud de los datos, L, como una función de la iteración de EM. La línea horizontal es el logaritmo de la verosimilitud del modelo verdadero. (a) Gráfico para el modelo de mezcla gaussiana de la Figura 20.8. (b) Gráfico para la red bayesiana de la Figura 20.10(a).

Una solución es situar distribuciones a priori en los parámetros del modelo, y aplicar la versión MAP de EM. Otra es reinicializar una componente con parámetros aleatorios nuevos si es muy pequeña o si está muy cercana a otra componente. También ayuda inicializar los parámetros con valores razonables.

Aprendizaje de redes bayesianas con variables ocultas

Para aprender una red bayesiana con variables ocultas, aplicamos las mismas ideas que funcionaban para mezclas de gaussianas. La Figura 20.10 representa una situación en la que hay dos bolsas de caramelos que han sido mezcladas. Los caramelos se describen por tres características: además del *Sabor* y el *Envoltorio*, algunos caramelos tienen un A*gujero* en el medio y otros no.

La distribución de los caramelos de cada bolsa se describe por un modelo de **Bayes simple**: las características son independientes, dada la bolsa, pero la distribución de probabilidades condicionales para cada característica depende de la bolsa. Los parámetros son los siguientes: θ es la probabilidad a priori de que un caramelo venga de la Bolsa 1; θ_{F1} y θ_{F2} son las probabilidades de que el sabor sea cereza, dado que el caramelo viene de la Bolsa 1 y la Bolsa 2 respectivamente; θ_{W1} y θ_{W2} son las probabilidades de que el envoltorio sea rojo; y θ_{H1} y θ_{H2} son las probabilidades de que el caramelo tenga un agujero. Observe que el modelo global es un modelo de mixtura. (De hecho, también podemos modelizar la mezcla de gaussianas como una red bayesiana, como muestra la Figura 20.10(b).) En la figura, la bolsa es una variable oculta, porque una vez que los caramelos se han mezclado no sabemos de qué bolsa venía cada uno. En este caso, ¿podemos recuperar las descripciones de las dos bolsas observando los caramelos a partir de la mezcla?

Figura 20.10 (a) Un modelo de mixtura para los caramelos. Las proporciones de sabores diferentes, envoltorios y número de agujeros dependen de la bolsa, que no se ha observado. (b) Red bayesiana para la mezcla gaussiana. La media y la covarianza de las variables observables **X** depende de la componente *C*.

Hagamos una iteración del algoritmo EM para este problema. Primero nos fijamos en los datos. Generamos 1.000 muestras a partir de un modelo cuyos verdaderos parámetros son

$$\theta = 0{,}5,\ \theta_{F1} = \theta_{W1} = \theta_{H1} = 0{,}8,\ \theta_{F2} = \theta_{W2} = \theta_{H2} = 0{,}3 \tag{20.7}$$

Es decir, es igualmente probable que los caramelos vengan de cualquier bolsa; la primera contiene en su mayoría caramelos de cereza con envoltorio rojo y agujero; la segunda contiene en su mayoría caramelos de lima con envoltorio verde y sin agujero. Los recuentos para los ocho tipos posibles de caramelos son los siguientes:

	$W = rojo$		$W = verde$	
	$H = 1$	$H = 0$	$H = 1$	$H = 0$
$F = cereza$	273	93	104	90
$F = lima$	79	100	94	167

Comenzamos inicializando los parámetros. Por simplicidad numérica elegimos[7]

$$\theta^{(0)} = 0{,}6,\ \theta^{(0)}_{F1} = \theta^{(0)}_{W1} = \theta^{(0)}_{H1} = 0{,}6,\ \theta^{(0)}_{F2} = \theta^{(0)}_{W2} = \theta^{(0)}_{H2} = 0{,}4 \tag{20.8}$$

Primero, trabajamos sobre el parámetro θ. En el caso completamente observable, podríamos estimar éste directamente a partir de los números *observados* de caramelos de las bolsas 1 y 2. En lugar de esto, ya que la bolsa es una variable oculta, calculamos los números *esperados*. El número esperado es la suma, sobre todos los caramelos, de la probabilidad de que el caramelo venga de la bolsa 1:

$$\theta^{(1)} = \hat{N}(Bolsa = 1)/N = \sum_{j=1}^{N} P(Bolsa = 1 | sabor_j, envoltorio_j, agujeros_j)/N$$

Estas probabilidades se pueden calcular utilizando cualquier algoritmo de inferencia para redes bayesianas. Para un modelo de Bayes simple como el de nuestro ejemplo, podemos hacer la inferencia «a mano», utilizando la regla de Bayes y aplicando independencia condicional:

$$\theta^{(1)} = \frac{1}{N}\sum_{j=1}^{N} \frac{P(sabor_j|Bolsa = 1)P(envoltorio_j|Bolsa = 1)P(agujeros_j|Bolsa = 1)P(Bolsa = 1)}{\Sigma_i\, P(sabor_j|Bolsa = i)P(envoltorio_j|Bolsa = i)P(agujeros_j|Bolsa = i)P(Bolsa = i)}$$

(Observe que la constante de normalización también depende de los parámetros.) Aplicando esta fórmula a los 273 caramelos de cereza con envoltorio rojo y agujeros, obtenemos una contribución de

$$\frac{273}{1000} \cdot \frac{\theta^{(0)}_{F1}\theta^{(0)}_{W1}\theta^{(0)}_{H1}\theta^{(0)}}{\theta^{(0)}_{F1}\theta^{(0)}_{W1}\theta^{(0)}_{H1}\theta^{(0)} + \theta^{(0)}_{F2}\theta^{(0)}_{W2}\theta^{(0)}_{H2}(1-\theta^{(0)})} \approx 0{,}22797$$

Continuando con los otros siete tipos de caramelos en la tabla de recuentos, obtenemos $\theta^{(1)} = 0{,}6124$.

[7] En la práctica es mejor elegirlos aleatoriamente, para evitar máximos locales debidos a simetría.

Ahora consideramos los otros parámetros, como θ_{F1}. En el caso completamente observable, estimaríamos éste directamente de los números *observados* de caramelos de cereza y lima de la bolsa 1. El número *esperado* de caramelos de cereza de la bolsa 1 viene dado por

$$\sum_{j:Sabor_j=cereza} P(Bolsa = 1|Sabor_j = cereza, envoltorio_j, agujeros_j)$$

Otra vez, estas probabilidades se pueden calcular mediante cualquier algoritmo de redes de Bayes. Completando este proceso, obtenemos los nuevos valores de todos los parámetros:

$$\begin{aligned} &\theta^{(1)} = 0{,}6124, \theta^{(1)}_{F1} = 0{,}6684, \theta^{(1)}_{W1} = 0{,}6483, \theta^{(1)}_{H1} = 0{,}6558 \\ &\theta^{(1)}_{F2} = 0{,}3887, \theta^{(1)}_{W2} = 0{,}3817, \theta^{(1)}_{H2} = 0{,}3827 \end{aligned} \tag{20.9}$$

El logaritmo de la verosimilitud de los datos se incrementa aproximadamente desde −2044, hasta alrededor de −2021 después de la primera iteración, como muestra la Figura 20.9(b). Es decir, la actualización mejora la verosimilitud con un factor de aproximadamente $e^{23} \approx 10^{10}$. En la décima iteración, el modelo aprendido está mejor ajustado que el modelo original ($L = -1.982{,}214$). De aquí en adelante, el proceso se hace muy lento. Esto no es extraño con EM, y muchos sistemas prácticos combinan EM con un algoritmo basado en el gradiente como Newton-Raphson (*véase* Capítulo 4) para la última fase del aprendizaje.

La lección general de este ejemplo es que *las actualizaciones de parámetros para aprendizaje en redes bayesianas con variables ocultas están directamente disponibles a partir de los resultados de la inferencia en cada ejemplo. Además, sólo se necesitan para cada parámetro las probabilidades a posteriori locales*. Para el caso general, en el que estamos aprendiendo los parámetros de las probabilidades condicionales para cada variable X_i, dados sus padres (esto es, $\theta_{ijk} = P(X_i = x_{ij}|Pa_i = pa_{ik})$) la actualización viene dada por los números esperados normalizados como sigue:

$$\theta_{ijk} \leftarrow \hat{N}(X_i = x_{ij}, Pa_i = pa_{ik})/\hat{N}(Pa_i = pa_{ik})$$

Los números esperados se obtienen sumando sobre los ejemplos, calculando las probabilidades $P(X_i = x_{ij}, Pa_i = pa_{ik})$ para cada uno, haciendo uso de cualquier algoritmo de inferencia de redes de Bayes. Para los algoritmos exactos (incluyendo eliminación de variables) todas estas probabilidades se pueden obtener directamente como resultado de hacer una inferencia estándar, sin la necesidad de cálculos extra específicos para el aprendizaje. Además, la información que se necesita para el aprendizaje está disponible *localmente* para cada parámetro.

Aprendizaje de modelos de Markov ocultos

Nuestra aplicación final de EM implica el aprendizaje de las probabilidades de transición en modelos de Markov ocultos (*hidden Markov models o HMMs*). Recordamos del Capítulo 15 que un modelo de Markov oculto se puede representar mediante una red de Bayes dinámica con una única variable de estado discreto, como se ilustra en la Figura 20.11.

Figura 20.11 Una red bayesiana dinámica expandida, que representa un modelo de Markov oculto (repetición de la Figura 15.14).

Cada punto de los datos consiste en una *secuencia* de observación de longitud finita, por lo que el problema es aprender las probabilidades de transición a partir de un conjunto de secuencias de observaciones (o posiblemente de una sola secuencia larga).

Ya hemos trabajado en el aprendizaje de redes de Bayes, pero hay una complicación: en las redes de Bayes cada parámetro es distinto; en un modelo de Markov oculto, las probabilidades individuales de transición desde el estado i hasta el estado j en el instante t, $\theta_{ijt} = P\ (X_{t+1} = j | X_t = i)$, se repiten a lo largo del tiempo, es decir, $\theta_{ijt} = \theta_{ij}$ para todo t. Para estimar las probabilidades de transición del estado i al estado j, simplemente calculamos la proporción *esperada* de veces que el sistema experimenta una transición al estado j cuando está en el estado i:

$$\theta_{ij} \leftarrow \sum_t \hat{N}(X_{t+1} = j, X_{t=i}) / \sum_t \hat{N}(X_{t=i})$$

Otra vez, los números esperados se calculan mediante algún algoritmo de inferencia HMM. El **algoritmo mixto** (*forward-backward*) mostrado en la Figura 15.4 se puede modificar muy fácilmente para computar las probabilidades necesarias. Un punto importante es que las probabilidades que se requieren son aquellas que se obtienen por **suavizado** en vez de por **filtrado**; esto es, necesitamos poner atención a la evidencia posterior en la estimación de la probabilidad de que ocurra una transición particular. Como dijimos en el Capítulo 15, la evidencia en un caso de asesinato normalmente se obtiene *después* de que el crimen (la transición del estado i al estado j) ocurra.

Forma general del algoritmo EM

Hemos visto varios ejemplos del algoritmo EM. Cada uno implica el cálculo de los valores esperados de las variable ocultas para cada ejemplo y después la actualización de los parámetros, utilizando los valores esperados como si fueran valores observados. Sea $\mathbf{x}$ el conjunto de todos los valores observados en todos los ejemplos, $\mathbf{Z}$ el conjunto de todas las variables ocultas para todos los ejemplos y θ el conjunto de todos los parámetros para el modelo probabilístico. El algoritmo EM es

$$\theta^{(i+1)} = \underset{\theta}{\operatorname{argmax}} \sum_{\mathbf{z}} P(\mathbf{Z} = \mathbf{z} | \mathbf{x}, \theta^{(i)}) L(\mathbf{x}, \mathbf{Z} = \mathbf{z} | \theta)$$

Esta ecuación resume el algoritmo EM. El paso-E es el cálculo del sumatorio, que es el valor esperado del logaritmo de la verosimilitud de los datos «completados» respecto a la distribución $P(\mathbf{Z} = \mathbf{z}|\mathbf{x}, \theta^{(i)})$, que es la distribución a posteriori de las variables ocultas, dados los datos. El paso-M es la maximización de este logaritmo de la verosimilitud esperado respecto a los parámetros. Para mezclas de gaussianas, las variables ocultas son Z_{ij}s, donde es Z_{ij} es 1 si el ejemplo j fue generado por la componente i. Para redes de Bayes, las variables ocultas son los valores de las variables no observadas de cada ejemplo. Para HMMs, las variables ocultas son las transiciones $i \rightarrow j$. Comenzando a partir de la forma general, es posible derivar un algoritmo EM para cada aplicación específica, una vez que se han identificado las variables ocultas apropiadas.

En cuanto entendamos la idea general de EM, será tarea fácil obtener todos los tipos de variantes y mejoras. Por ejemplo, en muchos casos el paso-E (el cálculo de las distribuciones a posteriori de las variables ocultas) es intratable, como en las redes de Bayes grandes. Puede ser que utilizando un paso-E *aproximado* se siga obteniendo un algoritmo de aprendizaje efectivo. Con un algoritmo de muestreo como MCMC (*véase* Apartado 14.5), el proceso de aprendizaje es muy intuitivo: cada estado (configuración de variables ocultas y observadas) visitado por MCMC se trata exactamente como si fuera una observación completa. Así, los parámetros se pueden actualizar directamente después de cada transición MCMC. Se ha demostrado que otras formas de inferencia aproximada, como los métodos con variaciones (*variational*) y para ciclos (*loopy*), son efectivas para el aprendizaje de redes muy grandes.

Aprendizaje de la estructura de las redes de Bayes con variables ocultas

En el Apartado 20.2, discutimos el problema del aprendizaje de las estructuras de redes de Bayes con datos completos. Cuando consideran variables ocultas, las cosas son más complicadas. En el caso más simple, las variables ocultas se listan junto con las variables observadas; de esta forma, aunque sus valores no son observados, el algoritmo de aprendizaje sabe que existen y debe encontrar un lugar para ellas en la estructura de la red. Por ejemplo, un algoritmo puede intentar aprender la estructura que se muestra en la Figura 20.7(a), dada la información de que *EnfermedadCorazon* (una variable de tres valores) debe ser incluida en el modelo. Si el algoritmo de aprendizaje no conociera esta información, habría dos elecciones posibles: fingir que los datos son realmente completos (lo que obligaría al algoritmo a aprender el modelo con muchos parámetros de la Figura 20.7(b)) o *inventar* nuevas variables ocultas para simplificar el modelo. El último enfoque se puede implementar incluyendo nuevas opciones de modificación en la estructura de búsqueda: además de modificar arcos, el algoritmo puede añadir o borrar una variable oculta o cambiar su aridad. Por supuesto, el algoritmo no sabrá que la nueva variable oculta que ha inventado se llama *EnfermedadCorazon*; ni tampoco tendrá nombres característicos para los valores. Afortunadamente, las nuevas variables ocultas inventadas normalmente estarán conectadas con variables que ya existían, y por lo tanto, un experto humano podrá inspeccionar las distribuciones condicionales locales en las que está implicada la nueva variable y averiguar su significado.

Como en el caso de datos completos, el aprendizaje puro de la estructura por máxima verosimilitud dará como resultado una red completamente conectada (además sin variables ocultas), por esto se requiere alguna forma de penalización de la complejidad. También podemos aplicar MCMC para aproximar el aprendizaje bayesiano. Por ejemplo, podemos aprender mezclas de gaussianas con un número de componentes desconocido muestreando sobre el número; la distribución a posteriori aproximada para el número de gaussianas viene dada por las frecuencias de muestreo del proceso MCMC.

Hasta ahora, el proceso del que hemos hablado tiene un bucle externo que es un proceso de búsqueda de la estructura, y un bucle interno que es un proceso de optimización paramétrica. Para el caso de datos completos, el bucle interno es muy rápido: extraer las frecuencias condicionales del conjunto de datos. Cuando hay variables ocultas, el bucle interno puede suponer muchas iteraciones de EM o de un algoritmo basado en el gradiente, y cada iteración implica el cálculo de las distribuciones a posteriori en una red de Bayes, que en sí mismo es un problema NP-duro. Hasta la fecha, se ha probado que este enfoque es impracticable para modelos de aprendizaje complejos. Una posible mejora es el algoritmo denominado **EM estructural**, que opera en casi todo de la misma forma que el ordinario (paramétrico), excepto que puede actualizar tanto la estructura como los parámetros. Al igual que el EM ordinario, utiliza los parámetros actuales para calcular los números esperados en el paso-E y después aplica esos números en el paso-M para elegir los nuevos parámetros, el EM estructural utiliza la estructura actual para calcular los números esperados y después aplica esos números en el paso-M para evaluar la verosimilitud de las nuevas estructuras potenciales. (Esto contrasta con el método del bucle externo/interno, que calcula los nuevos números esperados para cada estructura potencial.) De esta forma, el EM estructural puede realizar varias alteraciones estructurales en la red sin recalcular los números esperados, y es capaz de aprender estructuras de redes de Bayes no triviales. Sin embargo, todavía queda por hacer mucho trabajo antes de que podamos decir que el problema de aprendizaje de la estructura está resuelto.

EM ESTRUCTURAL

20.4 Aprendizaje basado en instancias

Hasta ahora, nuestra discusión sobre el aprendizaje estadístico se ha centrado principalmente en el ajuste de los parámetros de una familia *restringida* de modelos de probabilidad a un conjunto de datos *no restringido*. Por ejemplo, el agrupamiento no supervisado que utiliza mezclas de gaussianas supone que los datos se explican mediante la *suma* de un número *fijo* de distribuciones gaussianas. Denominamos a este tipo de métodos, métodos de **aprendizaje paramétrico**. Los métodos de aprendizaje paramétrico normalmente son simples y efectivos, pero frecuentemente, asumir una familia de modelos particular y restringida simplifica demasiado lo que está pasando en el mundo real, del que vienen los datos. Ahora, es cierto que cuando tenemos un conjunto de datos muy pequeño, no podemos esperar aprender un modelo complejo y detallado, pero parece absurdo mantener la hipótesis de complejidad fija cuando el conjunto de datos crece mucho.

APRENDIZAJE PARAMÉTRICO

APRENDIZAJE NO PARAMÉTRICO

APRENDIZAJE BASADO EN INSTANCIAS

En contraste con el aprendizaje paramétrico, los métodos de **aprendizaje no paramétrico** permiten que la complejidad de la hipótesis crezca con los datos. Cuantos más datos tengamos, más compleja puede ser la hipótesis. Estudiaremos dos familias muy sencillas de métodos de **aprendizaje basado en instancias** (o **aprendizaje basado en memoria**) no paramétricos, que se denominan así porque construyen hipótesis directamente a partir de las propias instancias de entrenamiento.

Modelos de vecinos más cercanos

VECINOS MÁS CERCANOS

La idea clave de los **modelos de vecinos más cercanos** es que es probable que las propiedades de un punto de entrada particular **x** sean similares a las de los puntos cercanos a **x**. Por ejemplo, si queremos hacer **estimación de la densidad** (esto es, estimar el valor de una densidad de probabilidad desconocida en **x**) podemos simplemente medir la densidad con aquellos puntos que están dispersos en las cercanías o en la vecindad de **x**. Esto parece muy sencillo, hasta que nos demos cuenta de que necesitamos especificar exactamente que queremos decir con «vecindad». Si la vecindad es muy pequeña, no contendrá ningún punto de los datos; si es muy grande, puede incluir a *todos* los puntos de los datos, dando lugar a una estimación de la densidad que es la misma en cualquier lugar. Una solución es definir la vecindad lo suficientemente grande como para incluir k puntos, donde k es lo suficientemente grande como para asegurar una estimación con significado. Para k fijo, el tamaño de la vecindad varía; si los datos están dispersos, la vecindad es grande, pero si los datos están muy juntos, la vecindad es pequeña. La Figura 20.12(a) muestra un ejemplo en dos dimensiones para datos dispersos. La Figura 20.13 muestra los resultados de la estimación de la densidad con *k vecinos*

Figura 20.12 (a) Un subconjunto de 128 puntos de los datos mostrados en la Figura 20.8(a), junto con dos puntos consulta y sus 10 vecinos más cercanos. (b) Un gráfico 3-D de la mezcla de gaussianas a partir de la que los datos fueron generados.

Figura 20.13 Estimación de la densidad utilizando *k*-vecinos-más-cercanos, aplicada a los datos de la Figura 20.12(a), para $k = 3$, 10 y 40 respectivamente.

más cercanos para esos datos, con $k = 3$, 10 y 40 respectivamente. Para $k = 3$, la densidad estimada en cualquier punto se basa únicamente en tres puntos cercanos y es altamente variable. Para $k = 10$, la estimación proporciona una buena reconstrucción de la verdadera densidad mostrada en la Figura 20.12(b). Para $k = 40$ la vecindad se hace muy grande y la estructura de los datos se pierde. En la práctica, utilizar un valor de k entre 5 y 10 proporciona buenos resultados para la mayoría de los conjuntos de datos con pocas dimensiones. También se puede elegir un buen valor de k utilizando validación cruzada.

Para identificar los vecinos más cercanos de un punto, necesitamos una métrica para medir la distancia, $D(\mathbf{x}_1, \mathbf{x}_2)$. El ejemplo de dos dimensiones de la Figura 20.12 utiliza la distancia Euclídea. Esto no es apropiado cuando cada dimensión del espacio se mide de forma diferente (por ejemplo altura y peso) porque cambiar la escala de una dimensión podría cambiar el conjunto de vecinos más cercanos. Una solución es estandarizar la escala para cada dimensión. Para hacer esto, medimos la desviación estándar de cada característica sobre el conjunto de datos y expresamos los valores de la característica como múltiplos de la desviación estándar de esa característica. (Esto es un caso especial de la **distancia de Mahalanobis**, que tiene en cuenta también la covarianza de las características.) Finalmente, para características discretas podemos utilizar la **distancia de Hamming**, que define $D(\mathbf{x}_1, \mathbf{x}_2)$ como el número de características en las que $\mathbf{x}_1$ y $\mathbf{x}_2$ difieren.

DISTANCIA MAHALANOBIS

DISTANCIA HAMMING

Las estimaciones de densidad, como las mostradas en la Figura 20.13, definen distribuciones conjuntas sobre el espacio de entrada. Sin embargo, al contrario que las redes bayesianas, una representación basada en instancias no contiene variables ocultas, lo que significa que no podemos llevar a cabo aprendizaje no supervisado como hacíamos con el modelo de mezcla de gaussianas. Todavía podemos utilizar la estimación de densidad para predecir un valor principal, y, dados los valores de entrada de las características $\mathbf{x}$, calculando $P(y|\mathbf{x}) = P(y, \mathbf{x})/P(\mathbf{x})$, siempre que los datos de entrenamiento incluyan valores para la característica principal.

También es posible utilizar la idea de vecinos más cercanos para hacer aprendizaje supervisado. Dado un ejemplo de test con entrada $\mathbf{x}$, la salida $y = h(\mathbf{x})$ se obtiene a partir de los valores-y de los k vecinos más cercanos a $\mathbf{x}$. En el caso discreto, se puede ob-

tener una predicción simple mediante el voto de la mayoría. En el caso continuo, podemos calcular la media de k valores o hacer regresión lineal, ajustando un hiperplano a k puntos y prediciendo el valor de $\mathbf{x}$ de acuerdo con el hiperplano.

El algoritmo de k-vecinos-más-cercanos es muy sencillo de implementar, requiere poco para ponerse a punto y normalmente tiene buen rendimiento. Es recomendable probarlo primero con un nuevo problema de aprendizaje. Sin embargo, para conjuntos de datos grandes, se requiere un mecanismo eficiente para encontrar los vecinos más cercanos de un punto de consulta $\mathbf{x}$ (calcular simplemente la distancia a cada punto podría llevar mucho tiempo). Se han propuesto varios métodos ingeniosos que preprocesan los datos de entrenamiento para que este paso sea eficiente. Desafortunadamente, la mayoría de estos métodos no escalan bien la dimensión del espacio (es decir, el número de características).

Los espacios de muchas dimensiones plantean un problema adicional, ¡los vecinos más cercanos en estos espacios están normalmente lejos! Considere un conjunto de datos de tamaño N en el hipercubo unidad d-dimensional y suponga una vecindad hipercúbica de lado b y volumen b^d. (El mismo argumento funciona para hiperesferas, pero la fórmula del volumen de una hiperesfera es más complicada.) Para contener k puntos, la vecindad media debe ocupar una fracción k/N del volumen total, que es 1. Por lo tanto, $b^d = k/N$, o $b = (k/N)^{1/d}$. De momento bastante bien. Ahora pongamos que el número de características d es 100, k es 10 y N es 1.000.000. Entonces, tenemos $b \approx 0{,}89$, esto es, ¡la vecindad se tiene que expandir a casi todo el espacio de entrada! Esto sugiere que los métodos de vecinos más cercanos no son de confianza para datos con muchas dimensiones. En pocas dimensiones no hay problema; con $d = 2$ tenemos $b = 0{,}003$.

Modelos núcleo

MODELO NÚCLEO

FUNCIÓN NÚCLEO

En un **modelo núcleo** (*kernel model*), vemos cada instancia de entrenamiento como si generara una pequeña función de densidad (una **función núcleo**) de sí misma. La estimación de densidad completa es exactamente la suma normalizada de todas las pequeñas funciones núcleo. Una instancia de entrenamiento $\mathbf{x}_i$ genera una función núcleo $K(\mathbf{x}, \mathbf{x}_i)$ que asigna una probabilidad a cada punto $\mathbf{x}$ del espacio. Así, la estimación de densidad es

$$P(\mathbf{x}) = 1\backslash N \sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i)$$

La función núcleo normalmente depende sólo de la *distancia* $D(\mathbf{x}, \mathbf{x}_i)$ de $\mathbf{x}$ a la instancia $\mathbf{x}_i$. La función núcleo más popular es (por supuesto) la gaussiana. Por simplicidad, asumiremos gaussianas esféricas con desviación estándar w a lo largo de cada eje, es decir,

$$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{(\omega^2 \sqrt{2\pi})^d} e^{-\frac{D(\mathbf{x}, \mathbf{x}_i)^2}{2\omega^2}}$$

donde d es el número de dimensiones de $\mathbf{x}$. Aún tenemos el problema de elegir un valor de w adecuado; como antes, hacer la vecindad muy pequeña proporciona una esti-

mación muy puntiaguda, *véase* Figura 20.14(a). En (b), un valor medio para w proporciona una reconstrucción muy buena. En (c), una vecindad muy grande da como resultado la pérdida de la estructura. Se puede elegir un buen valor de w utilizando validación cruzada.

El aprendizaje supervisado con núcleos se lleva a cabo tomando una combinación *ponderada* de *todas* las predicciones, a partir de las instancias de entrenamiento. (Compare esto con la predicción de k vecinos más cercanos, que toma una combinación no ponderada de las k instancias más cercanas.) El peso de la i-ésima instancia para un punto de consulta $\mathbf{x}$, viene dado por el valor del núcleo $K(\mathbf{x}, \mathbf{x}_i)$. Para una predicción discreta, tomamos un voto ponderado; para una predicción continua, tomamos la media ponderada o una regresión lineal ponderada. Observe que hacer predicciones con núcleos requiere observar *cada* instancia de entrenamiento. Es posible combinar núcleos con vecinos más cercanos indexando los esquemas para conseguir predicciones ponderadas únicamente a partir de las instancias más cercanas.

20.5 Redes neuronales

Una **neurona** es una célula del cerebro cuya función principal es la recogida, procesamiento y emisión de señales eléctricas. La Figura 1.2 mostraba un diagrama esquemático de una neurona típica. Se piensa que la capacidad de procesamiento de información del cerebro proviene principalmente de *redes* de este tipo de neuronas. Por esta razón, algunos de los primeros trabajos en IA pretendían crear **redes neuronales** artificiales. (Otros nombres para este campo son **conexionismo, procesamiento distribuido paralelo**, y **computación neuronal**.) La Figura 20.15 muestra un modelo matemático sencillo para la neurona ideada por McCulloch y Pitts (1943). Desde un punto de vista informal, la neurona se «dispara» cuando una combinación lineal de sus entradas excede un determinado umbral. Desde 1943 se han desarrollado modelos más realistas y detallados, tanto para neuronas como para sistemas más grandes del cerebro, llevándonos al campo moderno de la **neurociencia computacional**. Por otro lado, los investigadores de IA y de estadística se han interesado en las propiedades más abstractas de las re-

REDES NEURONALES

NEUROCIENCIA COMPUTACIONAL

Figura 20.14 Estimación núcleo de densidad para los datos de la Figura 20.12(a), utilizando núcleos gaussianos con $w = 0{,}02$, 0,07, y 0,20 respectivamente.

des neuronales, tales como su habilidad para desarrollar computación distribuida, para tolerar el ruido en la entrada, y para el aprendizaje. Aunque ahora comprendemos que otras clases de sistemas (incluyendo redes bayesianas) tienen estas propiedades, las redes neuronales permanecen como una de las formas más populares y efectivas de construir sistemas de aprendizaje y son dignas de estudio por sí mismas.

Unidades en redes neuronales

UNIDADES

CONEXIONES

ACTIVACIÓN

PESO

Las redes neuronales están compuestas de nodos o **unidades** (*véase* Figura 20.15) conectadas a través de **conexiones** dirigidas. Una conexión de la unidad j a la unidad i sirve para propagar la **activación** a_j de j a i. Además cada conexión tiene un **peso** numérico $W_{j,i}$ asociado, que determina la fuerza y el signo de la conexión. Cada unidad i primero calcula una suma ponderada de sus entradas:

$$in_i = \sum_{j=0}^{n} W_{j,i} a_j$$

FUNCIÓN DE ACTIVACIÓN

Luego aplica una **función de activación** g a esta suma para producir la salida:

$$a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right) \tag{20.10}$$

PESO DE SESGO

Nótese que se ha incluido un **peso de sesgo** $W_{0,i}$ conectado a una entrada fija $a_0 = -1$. Se explicará por qué en un momento.

La función de activación g se diseña con dos objetivos. Primero, queremos que la unidad esté «activa» (cercana a +1) cuando se proporcionen las entradas «correctas», e «inactiva» (cercana a 0) cuando se den las entradas «erróneas». Segundo, la activación tiene que ser no *lineal*, en otro caso la red neuronal en su totalidad se colapsaría con una sencilla función lineal (*véase* Ejercicio 20.17). En al Figura 20.16 se muestran dos posibles funciones g: la función **umbral** y la **función sigmoide** (también conocida como **función logística**). La función sigmoide tiene la ventaja de ser diferenciable, lo cual, como veremos más adelante, es importante para el algoritmo del aprendizaje de los pesos. Nó-

UMBRAL

FUNCIÓN SIGMOIDE

FUNCIÓN LOGÍSTICA

Figura 20.15 Un modelo matemático sencillo para una neurona. La activación de la salida de la unidad es $a_i = g(\sum_{j=0}^{n} W_{j,i} a_j)$, donde a_j es la activación de la salida de la unidad j y $W_{j,i}$ es el peso de la conexión de la unidad j a esta unidad.

Figura 20.16 (a) La función de activación umbral, con salida 1 cuando la entrada es positiva y 0 en otro caso. (Algunas veces se usa en lugar de esta la función **signo**, con salida + 1 dependiendo de la señal de la entrada.) (b) La función **sigmoide** $1/(1 + e^{-x})$.

tese que ambas funciones tienen un umbral (ni severo ni suave) de cero; los pesos de sesgo $W_{0,i}$ constituyen el umbral *real* de la unidad, en el sentido de que la unidad se activa cuando la suma de los pesos de las entradas «reales» $\Sigma_{j=1}^{n} W_{j,i}\, a_j$ (es decir, incluyendo la entrada del sesgo) excede $W_{0,i}$.

Podemos asemejar la operación de las unidades individuales a puertas lógicas. Una de las motivaciones originales para el diseño de unidades individuales (McCulloch y Pitts, 1943) fue su habilidad para representar funciones booleanas básicas. La Figura 20.17 muestra cómo se pueden representar las funciones booleanas AND, OR y NOT con unidades umbral con los pesos adecuados. Esto es importante ya que significa que podemos usar estas unidades para construir una red que calcule cualquier función booleana de las entradas.

Estructuras de las redes

REDES CON ALIMENTACIÓN-HACIA-DELANTE

REDES RECURRENTES

Existen dos categorías principales de estructuras de redes neuronales: acíclicas o **redes con alimentación-hacia-delante** y cíclicas o **redes recurrentes**. Una red con alimentación-hacia-delante representa una función de sus entradas actuales; de este modo, no tiene otro estado interno que no sea el de sus propios pesos. Por otro lado, una red re-

Figura 20.17 Las unidades con una función de activación umbral pueden actuar como puertas lógicas, con entradas y pesos de sesgo apropiados.

currente permite que sus salidas alimenten sus propias entradas. Esto significa que los niveles de activación de la red forman un sistema dinámico que debe alcanzar un estado estable, exhibir oscilaciones o incluso un comportamiento caótico. Además, la respuesta de la red dadas unas entradas depende de su estado inicial, que dependerá de entradas previas. Por lo tanto, las redes recurrentes (a diferencia de las redes con alimentación hacia delante) pueden tener memoria a corto plazo. Esto las hace más interesantes como modelos del cerebro, pero también más difíciles de entender. Esta sección se concentrará en las redes con alimentación hacia delante: al final del capítulo se proporcionan algunas lecturas adicionales sobre redes recurrentes.

Analicemos más detalladamente la afirmación de que una red con alimentación-hacia-delante representa una función de sus entradas. Consideremos la sencilla red mostrada en la Figura 20.18, que tiene dos unidades de entrada, dos **unidades ocultas** y una unidad de salida. (Para mantener simplicidad, se han omitido las unidades de sesgo en el ejemplo.) Dado un vector de entrada $\mathbf{x} = (x_1, x_2)$, las activaciones de las unidades de entrada se ponen a $(a_1, a_2) = (x_1, x_2)$ y la red calcula

UNIDADES OCULTAS

$$a_5 = g(W_{3,5}a_3 + W_{4,5}a_4) = g(W_{3,5}\, g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}\, g(W_{1,4}a_1 + W_{2,4}a_2)) \quad (20.11)$$

Es decir, expresando la salida de cada unidad oculta como una función de *sus* entradas, hemos mostrado la salida de la red como una suma, a_5, en función de las entradas de la red. Además, observamos que los pesos de la red actúan como *parámetros* de la función; escribiendo $\mathbf{W}$ para los parámetros, la red calcula $h_{\mathbf{W}}(\mathbf{x})$. Ajustando los pesos, cambiamos la función que representa la red. Esta es la manera en que se produce el aprendizaje en las redes neuronales.

Una red neuronal se puede usar para clasificación o para regresión. Para clasificaciones booleanas con entradas continuas (por ejemplo, con unidades sigmoides), es tradicional tener una única unidad de salida, con un valor por encima de 0,5 interpretado como una clase y con un valor por debajo de 0,5 como otra. Para clasificación en k-clases, se puede dividir el rango de la unidad de salida en k partes, pero es más común tener k unidades de salida separadas, donde el valor de cada una representa la verosimilitud relativa de esa clase dada una entrada actual.

Las redes neuronales con alimentación hacia delante normalmente se organizan en **capas**, de forma que cada unidad recibe entradas únicamente de las unidades de la capa que la precede inmediatamente. En las dos secciones siguientes mostraremos redes de

CAPAS

Figura 20.18 Una red neuronal muy sencilla con dos entradas, una capa oculta de dos unidades y una salida.

una única capa, sin unidades ocultas, y redes multicapa, con una o más capas de unidades ocultas.

Redes neuronales de una sola capa con alimentación hacia delante (perceptrones)

RED NEURONAL DE UNA SOLA CAPA

PERCEPTRÓN

Una red con todas las entradas conectadas directamente a las salidas se denomina **red neuronal de una sola capa**, o red **perceptrón**. Ya que cada unidad de salida es independiente de las otras (cada peso afecta sólo a una de las salidas) podemos limitar nuestro estudio a los perceptrones con una única unidad de salida, como el que se muestra en la Figura 20.19(a).

Comencemos examinado el espacio de hipótesis que un perceptrón puede representar. Con una función de activación umbral, el perceptrón puede representar una función booleana. Además de las funciones booleanas elementales AND, OR, y NOT (Figura 20.17), un perceptrón puede representar algunas funciones booleanas un poco más «complejas» de forma compacta. Por ejemplo, la **función mayoría**, cuya salida es 1 sólo si más de la mitad de sus n entradas están a 1, puede representarse con un perceptrón con peso $W_j = 1$ y umbral $W_0 = n/2$. Un árbol de decisión necesitaría $O(2^n)$ nodos para representar esta función.

Desafortunadamente, existen muchas funciones booleanas que el perceptrón umbral no puede representar. Mirando la Ecuación (20.10), observamos que el perceptrón umbral devuelve 1 si y sólo si la suma ponderada de sus entradas (incluyendo los sesgos) es positiva:

$$\sum_{j=0}^{n} W_j x_j > 0 \quad \text{o} \quad \mathbf{W} \cdot \mathbf{x} > 0$$

Figura 20.19 (a) Una red perceptrón consistente en tres unidades de salida perceptrón que comparten cinco entradas. Observando una unidad de salida particular (la segunda, con un círculo en negro), vemos que los pesos de sus conexiones de entrada no tienen efecto en las otras unidades de salida. (b) Un gráfico de la salida del perceptrón de dos unidades de entrada con una función de activación sigmoide.

La ecuación $\mathbf{W} \cdot \mathbf{x} = 0$ define un *hiperplano* en el espacio de entrada, así que el perceptrón devuelve 1 si y sólo si la entrada está en un lado de ese hiperplano. Por esta razón, el perceptrón umbral se denomina **separador lineal**. La Figura 20.20(a) y (b) muestra el hiperplano (una recta, en dos dimensiones) para la representación mediante un perceptrón de las funciones AND y OR de dos entradas. Los puntos negros indican un punto del espacio de entrada donde el valor de la función es 1, y los puntos blancos indican un punto donde el valor es 0. El perceptrón puede representar estas funciones porque existe una recta que separa todos los puntos blancos de todos los puntos negros. A estas funciones se las denomina **linealmente separables**. La Figura 20.20(c) muestra un ejemplo de una función que *no* es linealmente separable: la función XOR. Claramente, no hay manera de que el perceptrón umbral aprenda esta función. En general, *los perceptrones umbral pueden representar sólo funciones linealmente separables*. Éstas constituyen sólo una pequeña fracción de todas las funciones; el Ejercicio 20.14 pide cuantificar esta fracción. Los perceptrones sigmoides tienen una limitación similar, en el sentido de que representan sólo separadores lineales «suaves». (*Véase* Figura 20.19(b).)

SEPARADOR LINEAL

LINEALMENTE SEPARABLES

A pesar de su poder de expresividad limitado, los perceptrones umbral tienen algunas ventajas. En particular, *existe un algoritmo de aprendizaje sencillo que ajusta un perceptrón umbral a cualquier conjunto de entrenamiento que sea linealmente separable*. En vez de presentar este algoritmo, *obtendremos* un algoritmo muy relacionado para el aprendizaje en perceptrones sigmoides.

La idea del algoritmo, y en realidad de muchos algoritmos para aprendizaje de redes neuronales, es ajustar los pesos de la red para minimizar alguna medida del error que se produce con el conjunto de entrenamiento. Así, el aprendizaje se formula como una búsqueda optimizada en el **espacio de pesos**[8]. La medida «clásica» del error es la **suma de los errores cuadrados**, que usamos para regresión lineal en el Apartado 20.2. El error

ESPACIO DE PESOS

Figura 20.20 Separabilidad lineal en perceptrones umbral. Los puntos negros indican un punto del espacio de entrada donde el valor de la función es 1, y los puntos blancos indican un punto donde el valor es 0. El perceptrón devuelve 1 en la región del lado no sombreado de la línea. En (c), no existe una recta que clasifique correctamente las entradas.

[8] *Véase* el Apartado 4.4 para técnicas generales de optimización aplicables a espacios continuos.

cuadrado para un único ejemplo de entrenamiento con entrada **x** y valor verdadero de la salida y es

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2$$

donde $h_{\mathbf{W}}(\mathbf{x})$ es la salida del perceptrón para el ejemplo e y es el valor real de la salida.

Podemos usar el método del descenso del gradiente para reducir el error cuadrado calculando la derivada parcial de E con respecto a cada peso. Tenemos

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} \\ &= Err \times \frac{\partial}{\partial W_j} g\left(y - \sum_{j=0}^{n} W_j x_j\right) \\ &= -Err \times g'(in) \times x_j\end{aligned}$$

donde g' es la derivada de la función de activación[9]. En el algoritmo del descenso del gradiente, para *reducir* E, actualizamos los pesos de la siguiente manera:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j \tag{20.12}$$

donde α es la **tasa de aprendizaje**. Intuitivamente, esto tiene mucho sentido. Si el error $Err = y - h_{\mathbf{W}}(\mathbf{x})$ es positivo, la salida de la red es demasiado pequeña y por ello los pesos se *incrementan* para las entradas positivas y se *decrementan* para las entradas negativas. Cuando el error es negativo[10], ocurre lo contrario.

El algoritmo completo se muestra en la Figura 20.21. Introduce en la red los ejemplos de entrenamiento uno a uno, ajustando los pesos un poco después de cada ejemplo para reducir el error. Cada ciclo con todos de los ejemplos se denomina **época**. Las épocas se repiten hasta que se alcanza algún criterio de parada (típicamente, que los cambios de los pesos sean muy pequeños). Otros métodos calculan el gradiente para el conjunto total de entrenamiento añadiendo todas las contribuciones del gradiente en la Ecuación (20.12) antes de actualizar los pesos. El método del **gradiente estocástico** selecciona ejemplos aleatoriamente del conjunto de entrenamiento en vez de hacer ciclos con ellos.

ÉPOCA

GRADIENTE ESTOCÁSTICO

La Figura 20.22 muestra la curva de aprendizaje para un perceptrón en dos problemas diferentes. En la izquierda, se muestra la curva del aprendizaje de la función mayoría con 11 entradas Booleanas (es decir, la salida de la función es 1 si seis o más entradas son 1). Como esperaríamos, el perceptrón aprende la función rápidamente, ya que la función mayoría es linealmente separable. Por otro lado, el aprendizaje del árbol de decisión no progresa, ya que la función mayoría es muy difícil (aunque no imposible) de representar mediante un árbol de decisión. En la derecha, tenemos el ejemplo del restaurante. La solución del problema se representa fácilmente con un árbol de decisión,

[9] Para la sigmoide, la derivada viene dada por $g' = g(1 - g)$.

[10] Para los perceptrones umbral, donde $g'(in)$ no está definida, la **regla original de aprendizaje del perceptrón** desarrollada por Rosenblatt (1957) es idéntica a la Ecuación (20.12) excepto que se omite $g'(in)$. Ya que $g'(in)$ es la misma para todos los pesos, su omisión sólo cambia la magnitud y no la dirección de la actualización de todos los pesos para cada ejemplo.

función APRENDIZAJE-PERCEPTRÓN(*ejemplos*, *red*) **devuelve** perceptrón como hipótesis
  entrada: *ejemplos*, un conjunto de ejemplos, cada uno con entrada $\mathbf{x} = x_1, ..., x_n$ y salida y
    red, un perceptrón con pesos W_j, $j = 0 ... n$, y función de activación g

  repetir
    para cada e **en** *ejemplos* **hacer**
      $in \leftarrow (\Sigma_{j=0}^{n} W_j x_j[e]$
      $Err \leftarrow y[e] - g(in)$
      $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$
  hasta que se satisfaga algún criterio de parada
  devolver HIPÓTESIS-RED-NEURONAS(*red*)

Figura 20.21 El algoritmo de aprendizaje del descenso del gradiente para perceptrones, asumiendo una función de activación g diferenciable. Para perceptrones umbral, el factor $g'(in)$ se omite de la actualización de pesos. HIPÓTESIS-RED-NEURONA devuelve una hipótesis que calcula la salida de la red para cualquier ejemplo dado.

Figura 20.22 Comparación entre el rendimiento de los perceptrones y el de los árboles de decisión. (a) Los perceptrones son mejores en el aprendizaje de la función mayoría con 11 entradas. (b) Los árboles de decisión son mejores en el aprendizaje del predicado *Esperar* del ejemplo del restaurante.

pero no es linealmente separable. El mejor plano sólo clasifica correctamente el 65 por ciento de los datos.

Hasta aquí, hemos tratado los perceptrones como funciones determinísticas con salidas erróneas posibles. También es posible interpretar la salida de un perceptrón sigmoide como una *probabilidad*; específicamente, la probabilidad de que la salida verdadera sea 1 dadas las entradas. Con esta interpretación, se puede usar la sigmoide como una representación canónica para distribuciones condicionadas en redes bayesianas (*véase* Apartado 14.3). También se puede derivar una regla de aprendizaje usando el método estándar

de maximización del logaritmo de la verosimilitud de los datos, como se describió anteriormente en este capítulo. Veamos cómo funciona esto.

Consideremos un único ejemplo de entrenamiento con valor de salida verdadera T, y sea p la probabilidad devuelta por el perceptrón para este ejemplo. Si $T = 1$, la probabilidad condicional del dato es p, y si $T = 0$, la probabilidad condicional del dato es $(1 - p)$. Ahora podemos usar un truco sencillo para escribir el logaritmo de la verosimilitud de una forma que sea diferenciable. El truco es que una variable 0/1 en el *exponente* de una expresión actúa como una **variable indicadora**: p^T es p si $T = 1$ y 1 en otro caso; de forma análoga $(1 - p)^{(1-T)}$ es $(1 - p)$ si $T = 0$ y 1 en otro caso. Por ello, podemos escribir el logaritmo de la verosimilitud del dato como

VARIABLE INDICADORA

$$L = \log p^T (1 - p)^{(1-T)} = T \log p + (1 - T) \log(1 - p) \tag{20.13}$$

Gracias a las propiedades de la función sigmoide, el gradiente se reduce a una fórmula muy sencilla (Ejercicio 20.16):

$$\frac{\partial L}{\partial W_j} = Err \times a_j$$

Nótese que *el vector de actualización de los pesos para aprendizaje por máxima verosimilitud en los perceptrones sigmoides es esencialmente idéntico al vector de actualización para minimización del error cuadrado*. Por ello, podemos decir que los perceptrones tienen una interpretación probabilística incluso cuando la regla de aprendizaje se obtiene desde un punto de vista determinístico.

Redes neuronales multicapa con alimentación hacia delante

Ahora consideraremos redes con unidades ocultas. El caso más común supone una única capa oculta[11], como en la Figura 20.24. La ventaja de añadir capas ocultas es que se amplía el espacio de hipótesis que puede representar la red. Piense en cada unidad oculta como un perceptrón que representa una función umbral suave en el espacio de entradas, como se muestra en la Figura 20.19(b). Entonces, piense en una unidad de salida como una combinación lineal con umbral suave de varias de estas funciones. Por ejemplo, añadiendo dos funciones de umbral suave que se oponen y pasando un umbral al resultado, podemos obtener una función «cresta» como se muestra en la Figura 20.23(a). Combinando dos de estas crestas, haciendo un ángulo recto entre ellas (es decir, combinando la salida de cuatro unidades ocultas), obtenemos un «montículo» como se muestra en la Figura 20.23(b).

Con más unidades ocultas, podemos producir más montículos de diferentes tamaños en más lugares. De hecho, con una única capa oculta suficientemente grande, es posible representar cualquier función continua de las entradas con una precisión arbitraria;

[11] Algunas personas la denominan red de tres-capas, y algunas la denominan red de dos-capas (porque las entradas no son unidades «reales»). Evitaremos confusiones y la denominaremos «red de una única capa oculta».

Figura 20.23 (a) El resultado de combinar dos funciones de umbral suave que se oponen para producir una cresta. (b) El resultado de combinar dos crestas para producir un montículo.

con dos capas, incluso se pueden representar funciones discontinuas[12]. Desafortunadamente, para una estructura de red *determinada*, es difícil caracterizar exactamente qué funciones pueden ser representadas y cuáles no.

Suponga que queremos construir una red con una capa oculta para el problema del restaurante. Tenemos 10 atributos que describen cada ejemplo, así que necesitamos 10 unidades de entrada. ¿Cuántas unidades ocultas se necesitan? En la Figura 20.24, se muestra una red con cuatro unidades ocultas que resulta ser apropiada para este problema. El problema de escoger el número adecuado de unidades ocultas con antelación todavía no está bien comprendido. (*Véase* Apartado 20.5.)

Figura 20.24 Una red neuronal multicapa con una capa oculta y 10 entradas, adecuada para el problema del restaurante.

[12] La prueba es compleja, pero el punto principal es que el número requerido de unidades ocultas crece exponencialmente con respecto al número de entradas. Por ejemplo, se necesitan $2^n/n$ unidades ocultas para codificar todas las funciones booleanas de n entradas.

Los algoritmos de aprendizaje para redes multicapa son similares al algoritmo de aprendizaje del perceptrón mostrado en la Figura 20.21. Una pequeña diferencia es que podemos tener varias salidas, así que tenemos un vector de salida $\mathbf{h_W}(\mathbf{x})$ en vez de un único valor, y cada ejemplo tiene un vector de salida $\mathbf{y}$. La mayor diferencia es que, mientras que el error $\mathbf{y} - \mathbf{h_W}$ en la capa de salida es claro, el error en las capas ocultas no se conoce, porque los datos de entrenamiento no dicen cuál es el valor que han de tomar los nodos ocultos. Resulta que podemos **propagar hacia atrás** el error desde la capa de salida a las capas ocultas. El proceso de propagación-hacia-atrás proviene directamente a partir del gradiente del error total. Primero, describiremos el proceso con una justificación intuitiva; luego, veremos cómo se obtiene.

PROPAGAR HACIA ATRÁS

En la capa de salida, la regla de actualización de pesos es idéntica a la Ecuación (20.12). Tenemos múltiples unidades de salida, así será Err_i la i-ésima componente del vector de error $\mathbf{y} - \mathbf{h_W}$. También será conveniente definir un error modificado $\Delta_i = Err_i \times g'(in_i)$, de forma que la regla de actualización de los pesos se convierte en

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \tag{20.14}$$

Para actualizar las conexiones entre las unidades de entrada y las unidades ocultas necesitamos definir una cantidad análoga al término que representa el error para los nodos de salida. Aquí es donde podemos hacer la propagación hacia atrás del error. La idea es que el nodo oculto j es «responsable» de una fracción del error Δ_i en cada nodo de salida al que está conectado. Así, los valores de Δ_i se dividen de acuerdo con la fuerza de la conexión entre el nodo oculto y el nodo de salida, y se propagan hacia atrás para proporcionar los valores de Δ_j en la capa oculta. La regla de propagación para los valores de Δ es la siguiente**:**

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \tag{20.15}$$

Ahora la regla de actualización de los pesos entre la entradas y la capa oculta es casi idéntica a la regla de actualización de la capa de salida:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \,.$$

El proceso de propagación hacia atrás puede resumirse de la siguiente manera:

- Computar los valores Δ para las unidades de salida, usando el error observado.
- Comenzando con la capa oculta, repetir lo siguiente para cada capa en la red, hasta que se alcance la primera capa oculta:
  - — Propagar los valores Δ hacia atrás a la capa anterior.
  - — Actualizar los pesos entre las dos capas.

El algoritmo detallado se muestra en la Figura 20.25.

Para los aficionados a las matemáticas, obtendremos las ecuaciones de la propagación hacia atrás desde el principio. El error cuadrado para un único ejemplo se define como

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2$$

```
función APRENDIZAJE-PROP-ATRÁS(ejemplos, red) devuelve una red neuronal
  entrada: ejemplos, un conjunto de ejemplos, cada uno con vector de entrada x y vector de
           salida y red, una red multicapa con L capas, pesos W_{j,i}, función de activación g

  repetir
     para cada e en ejemplos hacer
        para cada nodo j en la capa de entrada hacer a_j ← x_j[e]
        para ℓ = 2 a M hacer
           in_i ← Σ_j W_{j,i} a_j
           a_i ← g (in_i)
        para cada nodo i en la capa de salida hacer
           Δ_i ← g'(in_i) × (y_i[e] − a_i)
        para ℓ = M − 1 a 1 hacer
           para cada nodo j en la capa ℓ hacer
              Δ_j ← g'( in_j) Σ_i W_{j,i} Δ_i
              para cada nodo i en la capa ℓ + 1 hacer
                 W_{j,i} ← W_{j,i} + α × a_j × Δ_i
  hasta que se satisfaga algún criterio de parada
  devolver HIPÓTESIS-RED-NEURONA(red)
```

Figura 20.25 El algoritmo de propagación hacia atrás para aprendizaje en redes multicapa.

donde la suma se realiza para todos los nodos de la capa de salida. Para obtener el gradiente respecto al peso específico $W_{j,i}$ en la capa de salida, necesitamos sólo desarrollar la activación a_i, ya que todos los otros términos del sumatorio no dependen de $W_{j,i}$:

$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) \\ &= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i\end{aligned}$$

con Δ_i definido como antes. Para obtener el gradiente respecto a los pesos $W_{k,j}$ que conectan la capa de entrada con la capa oculta, tenemos que mantener el sumatorio para todo i, ya que cada valor de salida a_i puede verse afectado por cambios en $W_{k,j}$. También tenemos que desarrollar las activaciones a_j. Mostraremos la derivación en más detalle, ya que es interesante ver cómo el operador de derivada se propaga hacia atrás por la red:

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} \\ &= -\sum_i (y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i}a_j\right) \\ &= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_i)}{\partial W_{k,j}}\end{aligned}$$

$$= -\sum_i \Delta_i W_{j,i}\, g'(in_j)\, \frac{\partial in_j}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i}\, g'(in_j) \frac{\partial}{\partial W_{k,j}} \left(\sum_k W_{k,j} a_k \right)$$

$$= -\sum_i \Delta_i W_{j,i}\, g'(in_j) a_k = -a_k \Delta_j$$

donde Δ_j se define como anteriormente. Así, conseguimos las reglas de actualización obtenidas anteriormente a partir de consideraciones intuitivas. Está también claro que el proceso puede continuarse para redes con más de una capa oculta, lo cual justifica el algoritmo general dado en la Figura 20.25.

Habiendo realizado todo el desarrollo matemático (o pasándolo por alto), veamos cómo se aplica una red con una única capa-oculta al problema del restaurante. En la Figura 20.26 mostramos dos curvas. La primera es una **curva de entrenamiento**, que muestra el error cuadrado medio para un conjunto de entrenamiento de 100 ejemplos del restaurante durante el proceso de actualización de pesos. Esto demuestra que la red en realidad converge a un ajuste perfecto de los datos de entrenamiento. La segunda curva es la curva de aprendizaje estándar para los datos del restaurante. La red neuronal aprende bien, aunque no tan rápido como el aprendizaje con árboles de decisión; esto quizá no es sorprendente, porque inicialmente los datos fueron generados a partir de un árbol de decisión sencillo.

CURVA DE ENTRENAMIENTO

Por supuesto, las redes neuronales son capaces de hacer tareas de aprendizaje más complejas, aunque se debe decir que es necesaria una cierta cantidad de trabajo para conseguir la estructura correcta de la red y para conseguir la convergencia a algo cercano al óptimo global del espacio de pesos. Existen literalmente decenas de miles de aplicaciones de redes neuronales publicadas. El Apartado 20.7 muestra una de estas aplicaciones más detalladamente.

Figura 20.26 (a) Curva de entrenamiento mostrando la reducción gradual del error a medida que los pesos se modifican a través de varias épocas, para un conjunto dado de ejemplos del dominio del restaurante. (b) Curvas de aprendizaje comparativas que muestran que el aprendizaje de árboles de decisión es levemente mejor que la propagación hacia atrás en una red multicapa.

Aprendizaje de la estructura de las redes neuronales

Hasta aquí, hemos considerado el problema del aprendizaje de los pesos, dada una estructura fija de la red; al igual que con redes bayesianas, también necesitamos comprender cómo encontrar la mejor estructura de la red. Si elegimos una red que es demasiado grande, será capaz de memorizar todos los ejemplos formando una gran tabla de búsqueda, pero no generalizará necesariamente bien para entradas que no se han visto anteriormente[13]. En otras palabras, como todos los modelos estocásticos, las redes neuronales son sujeto de **sobreajuste** cuando hay demasiados parámetros en el modelo. Vimos esto en la Figura 18.1 (Apartado 18.3), donde los modelos con muchos parámetros en (b) y en (c) se ajustan a todos los datos, pero no generalizarían tan bien como los modelos de pocos parámetros en (a) y (d).

Si nos centramos en redes totalmente conectadas, las únicas elecciones por las que nos podemos preocupar son el número de capas ocultas y su tamaño. El enfoque más usual es intentar varias y quedarnos con la mejor. Se necesitan las técnicas de **validación cruzada** del Capítulo 18 si queremos evitar el ***peeking*** del mejor conjunto. Es decir, elegimos la arquitectura de la red que proporciona la mayor precisión de predicción en los conjuntos de validación.

DAÑO CEREBRAL ÓPTIMO

Si queremos considerar redes que no están conectadas en su totalidad, necesitamos encontrar algún método de búsqueda efectivo a través del gran espacio de topologías de posibles conexiones. El algoritmo del **daño cerebral óptimo** (*optimal brain damage*) comienza con una red totalmente conectada y va eliminando conexiones. Después de que la red está entrenada por primera vez, un enfoque teórico de información identifica una selección óptima de las conexiones que pueden ser eliminadas. La red vuelve a entrenarse, y si su rendimiento no se ve decrementado se repite el proceso. Además de la eliminación de conexiones, también es posible eliminar unidades que no contribuyen mucho al resultado.

TILING

Se han propuesto varios algoritmos para conseguir una red más grande, aumentando una red más pequeña. Uno de ellos, el algoritmo «***Tiling***», se parece al aprendizaje de listas de decisión. La idea es comenzar con una única unidad que se comporta de la mejor forma posible para devolver la salida correcta para tantos ejemplos de entrenamiento como sea posible. Se añaden más unidades para corregir los ejemplos en los que la primera unidad falló. El algoritmo añade sólo las unidades necesarias para cubrir todos los ejemplos.

20.6 Máquinas núcleo

Nuestra discusión sobre redes neuronales nos ha dejado con un dilema. Las redes de una única capa tienen un algoritmo de aprendizaje más simple y eficiente, pero tienen un poder de expresividad muy limitado (sólo pueden aprender fronteras lineales de decisión en el espacio de entradas). Por otro lado, las redes multicapa son más expresivas (pue-

[13] Se ha observado que redes muy grandes *hacen* generalizaciones correctas *mientras que los pesos se mantienen pequeños*. Esta restricción mantiene los valores de activación en la región *lineal* de la función sigmoide $g(x)$ donde x está cercana a cero. Esto significa que la red se comporta como una función lineal (Ejercicio 20.17) con pocos parámetros.

MÁQUINA DE VECTORES SOPORTE

MÁQUINA NÚCLEO

den representar funciones no lineales generales) pero son muy difíciles de entrenar debido a la abundancia de mínimos locales y la gran dimensión del espacio de pesos. En esta sección, exploraremos una familia de métodos de aprendizaje, relativamente nuevos, denominados **máquinas de vectores soporte** (SVMs), o más generalmente, **máquinas núcleo**. Hasta cierto punto, las máquinas núcleo nos proporcionan lo mejor de ambos mundos. Es decir, estos métodos usan un algoritmo de aprendizaje eficiente *y* pueden representar funciones no lineales complejas.

El tratamiento completo de las máquinas núcleo está fuera del alcance de este libro, pero podemos ilustrar la idea principal a través de un ejemplo. La Figura 20.27(a) muestra un espacio de entrada bidimensional definido por atributos $\mathbf{x} = (x_1, x_2)$, con ejemplos positivos ($y = +1$) dentro de un círculo y ejemplos negativos ($y = -1$) fuera. Obviamente, no existe ningún separador lineal para este problema. Ahora, suponga que expresamos de nuevo los datos de entrada usando algunas características computadas, es decir, hacemos corresponder cada vector de entrada $\mathbf{x}$ con un nuevo vector de valores de características, $F(\mathbf{x})$. En particular, usaremos las tres características siguientes

$$f_1 = x_1^2, \qquad f_2 = x_2^2, \qquad f_3 = \sqrt{2}x_1x_2 \tag{20.16}$$

Veremos en breve de dónde vienen, pero ahora, veamos qué ocurre. La Figura 20.27(b) muestra los datos en el nuevo espacio tridimensional definido por las tres características; ¡los datos son *linealmente separables* en el espacio! Este fenómeno es normalmente bastante general: si los datos se hacen corresponder a un espacio con una dimensión suficientemente alta, serán siempre linealmente separables. Aquí, hemos usado sólo tres

Figura 20.27 (a) Un entrenamiento en dos dimensiones con ejemplos positivos como círculos negros y ejemplos negativos como círculos blancos. La frontera de la decisión verdadera, $x_1^2 + x_2^2 \leq 1$, también se muestra. (b) Los mismos datos después de hacer la correspondencia a un espacio de entrada de tres dimensiones ($x_1^2, x_2^2, \sqrt{2}x_1x_2$). La frontera de decisión circular en (a) se convierte en una frontera de decisión lineal en tres dimensiones.

dimensiones[14], pero si tenemos N puntos de los datos, excepto en casos especiales, serán siempre separables en un espacio de dimensión N-1 o mayor (Ejercicio 20.21).

Luego, ¿es así? ¿Únicamente tenemos que producir cantidades de características computadas y luego encontrar un separador lineal en el espacio correspondiente de alta dimensión? Desafortunadamente, no es tan fácil. Recuerde que un separador lineal en un espacio de d dimensiones se define a través de una ecuación con d parámetros, así que hay un serio peligro de sobreajuste de los datos si $d \approx N$, donde N es el número de puntos de los datos. (Esto es como el sobreajuste de datos con un polinomio de grado alto, que se discutió en el Capítulo 18.) Por esta razón, las máquinas núcleo normalmente encuentran el separador lineal *óptimo*, el que da lugar a un **margen** más grande entre él y los ejemplos positivos en un lado y los ejemplos negativos en el otro. (*Véase* la Figura 20.28.) Se puede mostrar, usando argumentos de la teoría computacional del aprendizaje (Apartado 18.5), que este separador tiene propiedades deseables en términos de generalización robusta para nuevos ejemplos.

MARGEN

Ahora, ¿cómo encontraremos este separador? Resulta que esto es un problema de optimización de **programación cuadrática**. Suponga que tenemos ejemplos $\mathbf{x}_i$ con clasificaciones $y_i = \pm 1$ y queremos encontrar un separador óptimo en el espacio de entrada; entonces el problema de programación cuadrática consiste en encontrar los valores del parámetro α_i que maximizan la expresión

PROGRAMACIÓN CUADRÁTICA

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{20.17}$$

Figura 20.28 Un plano, proyectado en la primera de la dos dimensiones, del separador óptimo mostrado en la Figura 20.27(b). El separador se muestra como una línea realzada, con los puntos más cercanos (los **vectores soporte**) marcados con círculos. El **margen** es la separación entre los ejemplos positivos y los ejemplos negativos.

[14] El lector habrá notado que podríamos haber usado sólo f_1 y f_2, pero la correspondencia en 3D ilustra mejor la idea.

sujeta a las restricciones $\alpha_i \geq 0$ y $\Sigma_i \alpha_i y_i = 0$. Aunque la derivada de esta expresión no es crucial para la historia, hace que tenga dos propiedades importantes. Primero, la expresión tiene un único máximo global que puede encontrarse de forma eficiente. Segundo, *los datos entran en la expresión sólo en forma de productos de pares de puntos*. Esta segunda propiedad es también verdadera en la ecuación para el propio separador; una vez que ha sido calculado el óptimo α_i, es

$$h(\mathbf{x}) = \text{signo}\left(\sum_i \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}_i)\right) \tag{20.18}$$

Una última propiedad importante del separador óptimo definido por esta ecuación es que los pesos α_i asociados con cada punto de los datos son *cero* excepto para aquellos puntos más cercanos al separador, los denominados **vectores soporte**. (Se denominan así porque «soportan» al plano separador.) Debido a que normalmente existen menos vectores soporte que puntos de los datos, el número efectivo de parámetros que define el separador óptimo es normalmente más pequeño que N.

VECTOR SOPORTE

Normalmente no esperaríamos encontrar un separador lineal en el espacio de entrada $\mathbf{x}$, pero es fácil ver que podemos encontrar separadores lineales en el espacio de características de alta dimensión $F(\mathbf{x})$ simplemente reemplazando $\mathbf{x}_i \cdot \mathbf{x}_j$ en la Ecuación (20.17) por $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$. Esto por sí mismo no es remarcable (reemplazar $\mathbf{x}$ por $F(\mathbf{x})$ en *cualquier* algoritmo de aprendizaje tiene el efecto que se necesita) pero, el producto tiene algunas propiedades especiales. Resulta que $F(\mathbf{x}_i) \cdot F(\mathbf{x}_j)$ a menudo puede operarse sin calcular primero F para cada punto. En nuestro espacio de características de tres dimensiones definido por la Ecuación (20.16), un poco de álgebra muestra que

$$F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

La expresión $(\mathbf{x}_i \cdot \mathbf{x}_j)^2$ se denomina una **función núcleo**, que normalmente se escribe como $K(\mathbf{x}_i, \mathbf{x}_j)$. En el contexto de las máquinas núcleo es una función que se puede aplicar a pares de datos de entrada para evaluar los productos en el espacio de características correspondiente. Así que, podemos restablecer la pretensión del principio de este párrafo de la siguiente manera: podemos encontrar separadores lineales en el espacio de características de alta dimensión $F(\mathbf{x})$ simplemente remplazando $\mathbf{x}_i \cdot \mathbf{x}_j$ en la Ecuación (20.17) por una función núcleo $K(\mathbf{x}_i, \mathbf{x}_j)$. Así, podemos aprender en el espacio de alta dimensión, pero calculamos sólo funciones núcleo en vez del conjunto total de características para cada punto de los datos.

El siguiente paso, que ahora debería ser obvio, es ver que no hay nada especial en el núcleo $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$. Corresponde a un espacio de características particular de dimensión alta, pero otras funciones núcleo corresponden a otros espacios de características. Un resultado venerable en matemáticas, el **teorema de Mercer** (1909), dice que cualquier función núcleo «razonable»[15] corresponde a *algún* espacio de características. Estos espacios de características pueden ser muy grandes, incluso para núcleos que parecen simples. Por ejemplo, el **núcleo polinómico**, $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i \cdot \mathbf{x}_j)^d$, corresponde a un espacio de características cuya dimensión es exponencial en d. Usando dichos

TEOREMA DE MERCER

NÚCLEO POLINÓMICO

[15] Aquí, «razonable» significa que la matriz $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ es definida positiva; consultar Apéndice A.

núcleos en la Ecuación (20.17), *los separadores lineales óptimos se pueden encontrar eficientemente en espacios de características con billones (o, en algunos casos, infinitamente más) de dimensiones*. Cuando los separadores lineales resultantes se proyectan de vuelta al espacio de entrada original, pueden corresponder a fronteras no lineales arbitrariamente onduladas entre los ejemplos positivos y negativos.

En la sección previa se mencionó que las máquinas núcleo destacan en el reconocimiento de dígitos escritos a mano: están adaptándose rápidamente para otras aplicaciones, especialmente para aquellas con muchas características de entrada. Como parte de este proceso, se han diseñado muchos núcleos nuevos para trabajar con cadenas, árboles y otros tipos de datos no numéricos. También se ha observado que los métodos núcleo se pueden aplicar no sólo con algoritmos de aprendizaje que encuentran separadores lineales óptimos, sino también con cualquier otro algoritmo que pueda reformularse para trabajar sólo con productos de pares de puntos de los datos, como en la Ecuación 20.17 y 20.18. Una vez que esto se hace, el producto se reemplaza por una función núcleo y tenemos una versión del algoritmo con **núcleos**. Esto se puede hacer fácilmente para el aprendizaje del *k*-vecinos-más-cercanos y para el aprendizaje del perceptrón, entre otros.

VERSIÓN CON NÚCLEOS

20.7 Caso de estudio: reconocedor de dígitos escritos a mano

Reconocer dígitos escritos a mano es un problema importante en muchas aplicaciones, incluyendo la ordenación automática de cartas por el código postal, la lectura automática de ingresos de cheques y de impuestos, y la entrada manual de datos en computadores. Es un área donde se han hecho progresos rápidos, en parte debido a algoritmos de aprendizaje mejores y en parte debido a la disponibilidad de mejores conjuntos de entrenamiento. El Instituto Nacional de Ciencia y Tecnología de Estados Unidos (**NIST**) ha archivado una base de datos de 60.000 dígitos etiquetados, cada uno de $20 \times 20 = 400$ píxeles con una escala de grises de 8-bits. Se ha convertido en uno de los problemas estándar de las herramientas para comparar nuevos algoritmos de aprendizaje. Algunos ejemplos de dígitos se muestran en la Figura 20.29.

Figura 20.29 Ejemplos de la base de datos NIST de dígitos escritos a mano. La fila de arriba: ejemplos de dígitos 0-9 que son fáciles de identificar. La fila de abajo: ejemplos más difíciles de los mismos dígitos.

Se han intentado distintos enfoques de aprendizaje. Uno de los primeros, y probablemente el más sencillo, es el clasificador de **3 vecinos más cercanos**, que también tiene la ventaja de que no requiere tiempo de entrenamiento. Un algoritmo basado en memoria, sin embargo, debe almacenar las 60.000 imágenes, y su tiempo de ejecución es lento. Se consiguió una tasa de error del 2,4 por ciento.

Para este problema se diseñó una **red neuronal con una única capa oculta** con 400 unidades de entrada (una por píxel) y 10 unidades de salida (una por clase). Usando validación-cruzada, se encontró que aproximadamente 300 unidades ocultas proporcionaban el mejor funcionamiento. Con total interconexión entre capas, hubo un total de 123.300 pesos. Esta red consiguió una tasa de error del 1,6 por ciento.

Una serie de **redes neuronales especializadas** denominadas LeNet fueron ideadas para aprovechar la estructura del problema (que la entrada consiste en píxeles de un vector bidimensional, y que pequeños cambios en la posición o en la inclinación de la imagen carecen de importancia). Cada red tiene una capa de entrada de 32×32 unidades, sobre las cuales los 20×20 píxeles se centraron tal que cada unidad de entrada se presenta con una vecindad local de píxeles. Esto se continuó con tres capas de unidades ocultas. Cada capa consistía en varios niveles de $n \times n$ vectores, donde cada n es más pequeño que el de la capa previa, de forma que la red va reduciendo la entrada, y se obliga a que los pesos de cada unidad en un nivel sean idénticos, de manera que el nivel está actuando como un detector de características: puede distinguir una característica tal como una línea vertical larga o un arco semi-circular corto. La capa de salida tiene 10 unidades. Se probaron muchas versiones de esta arquitectura; una representativa tenía capas ocultas con 768, 192 y 30 unidades, respectivamente. El conjunto de entrenamiento se aumentó aplicando transformaciones afines a las entradas reales: trasladando, rotando un poco y escalando la imagen. (Por supuesto, las transformaciones tienen que ser pequeñas, o sino ¡un 6 se transformará en un 9!) La mejor tasa de error conseguida por LeNet fue del 0,9 por ciento.

Una **red neuronal potenciada** (*boosted*) combina tres copias de la arquitectura de LeNet. La segunda se entrena a partir de una mezcla de patrones en los que la primera fallaba en un 50 por ciento, y la tercera se entrena con patrones para los cuales las dos primeras no estuvieron de acuerdo. Durante el test, las tres redes votaban con sus pesos por cada uno de los 10 dígitos, y las puntuaciones se sumaban para determinar el ganador. La tasa de error fue del 0,7 por ciento.

Una **máquina de vectores soporte** (*véase* el Apartado 20.6) con 25.000 vectores soporte consiguió una tasa de error del 1,1 por ciento. Esto es remarcable ya que la técnica SVM, al igual que el enfoque sencillo de vecinos-más-cercanos, no requiere casi esfuerzo o experimentos reiterados por parte del desarrollador, sin embargo se acercó al funcionamiento de LeNet, que supuso años de desarrollo. En realidad, las máquinas de vectores soporte no hacen uso de la estructura del problema, y actuarían de igual forma si los píxeles se presentaran en un orden permutado.

MÁQUINA VIRTUAL DE VECTORES SOPORTE

Una **máquina virtual de vectores soporte** comienza con una SVM regular y luego la mejora con una técnica que se diseña para aprovechar la estructura del problema. En vez de permitir productos de todos los pares de píxeles, este enfoque se concentra en los núcleos formados a partir de pares de píxeles cercanos. También aumenta el conjunto de entrenamiento con transformaciones de los ejemplos, como Le-

Net. Una SVM virtual consiguió la mejor tasa de error conseguida hasta la fecha, 0,56 por ciento.

El **encaje de formas** (*shape matching*) es una técnica de visión computacional que se usa para detectar las partes que se corresponden de dos imágenes diferentes de objetos. (*Véase* el Capítulo 24.) La idea es escoger un conjunto de puntos en cada una de las dos imágenes, y luego calcular qué punto de la segunda imagen se corresponde con cada punto de la primera. A partir de esto, se calcula una transformación entre las imágenes. La transformación nos proporciona una medida de la distancia entre las imágenes. La medida de la distancia está mejor justificada que contar el número de píxeles diferentes, y resulta que un algoritmo de 3 vecinos más cercanos que usa esta medida de distancia funciona bastante bien. Entrenando con sólo 20.000 de los 60.000 dígitos, y usando 100 puntos de prueba por imagen, extraídos de un detector de bordes Canny, un clasificador de encaje de formas consigue una tasa de error del 0,63 por ciento.

Se estima que los **humanos** tienen una tasa de error del 0,2 por ciento aproximadamente en este problema. Esta cifra es algo sospechosa porque a los humanos no se les ha puesto a prueba tan intensamente como a los algoritmos de aprendizaje automático. En un conjunto similar de datos de dígitos del Servicio Postal de Estados Unidos, los errores de los humanos fueron de un 2,5 por ciento.

La siguiente figura resume las tasas de error, tiempo de desarrollo, memoria requerida y cantidad de tiempo de entrenamiento para los siete algoritmos que se han discutido. También añade otra medida, el porcentaje de dígitos que deben rechazarse para conseguir un error del 0,5 por ciento. Por ejemplo, si a la SVM se le permite rechazar el 1,8 por ciento de la entrada (es decir, dejar que otro sistema haga el juicio final) su tasa de error sobre el 98,2 por ciento restante de la entrada se reduce de 1,1 por ciento a 0,5 por ciento.

La siguiente tabla resume la tasa de error y algunas de las otras características de las siete técnicas discutidas.

	3 NN	300 ocultas	LeNet	LeNet potenciada	SVM	SVM Virtual	Encaje formas
Tasa de error (pct.)	2,4	1,6	0,9	0,7	1,1	0,56	0,63
T. Ejec. (msg/dígito)	1000	10	30	50	2000	200	
M. Requerida (Mbyte)	12	0,49	0,012	0,21	11		
T. entrenamiento (días)	0	7	14	30	10		
% rechazado para un error del 0,5%	8,1	3,2	1,8	0,5	1,8		

20.8 Resumen

Los métodos estadísticos de aprendizaje van desde el simple cálculo de medias hasta la construcción de modelos complejos tales como las redes bayesianas o las redes neuronales. Tienen aplicación en Informática, Ingeniería, Neurobiología, Psicología y Física. En este capítulo se han presentado algunas de las ideas básicas y se ha proporcionado una idea de la base matemática. Los principales puntos son los siguientes:

- Los métodos de **aprendizaje bayesiano** formulan el aprendizaje como una forma de inferencia probabilística, usando las observaciones para actualizar una distribución previa sobre las hipótesis. Este enfoque implementa bien la navaja de Ockham, pero se vuelve rápidamente intratable para espacios de hipótesis complejos.
- El aprendizaje **máximo *a posteriori*** (MAP) selecciona una única hipótesis más probable dados los datos. También se usa la hipótesis a priori y el método es a menudo más tratable que el aprendizaje Bayesiano completo.
- El aprendizaje de **máxima verosimilitud** simplemente selecciona la hipótesis que maximiza la verosimilitud de los datos; es equivalente al aprendizaje MAP con una distribución uniforme a priori. En casos sencillos tales como regresión lineal y redes bayesianas completamente observables, las soluciones de máxima verosimilitud pueden encontrarse fácilmente de una forma cerrada. El aprendizaje **bayesiano simple** es una técnica particularmente efectiva que se amplía bien.
- Cuando algunas variables están ocultas, las soluciones locales de máxima verosimilitud se pueden encontrar usando el algoritmo EM. Las aplicaciones incluyen agrupamiento usando mezclas de gaussianas, aprendizaje en redes bayesianas, y aprendizaje de modelos de Markov ocultos.
- El aprendizaje de la estructura de redes bayesianas es un ejemplo de **selección del modelo**. Esto normalmente requiere una búsqueda discreta en el espacio de las estructuras. Se requieren algunos métodos para llegar a un compromiso entre la complejidad del modelo y el grado de ajuste.
- Los **modelos basados en instancias** representan una distribución usando la colección de ejemplos de entrenamiento. Por ello, el número de parámetros crece con el conjunto de entrenamiento. Los métodos de **vecinos más cercanos** buscan la instancia más cercana al punto en cuestión, mientras que los métodos **núcleo** forman una combinación ponderada de las distancias entre todas las instancias.
- Las **redes neuronales** son funciones complejas no lineales con muchos parámetros. Sus parámetros pueden aprenderse a partir de datos con ruido y se han usado para miles de aplicaciones.
- Un **perceptrón** es una red neuronal con alimentación hacia delante sin unidades ocultas que puede representar sólo funciones **linealmente separables**. Si los datos son linealmente separables, se puede usar una regla de actualización de pesos sencilla para ajustar exactamente los datos.
- Las redes neuronales **multicapa con alimentación hacia delante** pueden representar cualquier función, con suficientes unidades. El algoritmo de **propagación hacia atrás** (*back-propagation*) implementa un descenso del gradiente en el espacio de parámetros para minimizar el error de la salida.

El aprendizaje estadístico continúa siendo un área de investigación muy activa. Se han hecho enormes avances tanto en la teoría como en la práctica, hasta el punto de que es posible aprender casi cualquier modelo para el cual sea posible inferencia aproximada o exacta.

NOTAS BIBLIOGRÁFICAS E HISTÓRICAS

La aplicación de técnicas estadísticas de aprendizaje en IA fue un área activa de investigación en los primeros años (*véase* Duda y Hart, 1973) pero se fue separando de la principal corriente de IA, ya que este campo se concentró en métodos simbólicos. Continuó en varias formas (algunas explícitamente probabilísticas, y otras no) en áreas tales como **reconocimiento de patrones** (Devroye *et al*., 1996) y **recuperación de información** (Salton y McGill, 1983). Un resurgimiento del interés ocurrió después de la introducción de los modelos de redes bayesianas a últimos de la década de los 80; aproximadamente al mismo tiempo, comenzó a surgir una visión estadística del aprendizaje de redes neuronales. A últimos de la década de los 90, hubo una convergencia notable de interés en el aprendizaje automático, en estadística, y en redes neuronales, centradas en métodos para la creación de grandes modelos probabilísticos a partir de los datos.

El modelo de Bayes simple es una de las formas más antigua y sencilla de redes bayesianas de la década de los 50. Sus orígenes, mencionados en las notas del final del Capítulo 13, están en parte explicados por Domingos y Pazzani (1997). Una forma *potenciada* del aprendizaje de Bayes simple ganó la primera copa KDD en la competición de minería de datos (Elkan, 1997). Heckerman (1998) proporciona una excelente introducción al problema general de aprendizaje de redes bayesianas. El aprendizaje de parámetros bayesianos con distribuciones Dirichlet *a priori* para redes bayesianas se discutió en Spiegelhalter *et al*. (1993). El paquete de software BUGS (Gilks *et al*., 1994) incorpora muchas de estas ideas y proporciona una herramienta muy potente para la formulación y el aprendizaje de modelos probabilísticos complejos. Los primeros algoritmos de aprendizaje de la estructura de una red bayesiana usaban tests de independencia condicional (Pearl, 1988; Pearl y Verma, 1991). Spirtes *et al*. (1993) desarrolló un enfoque comprensivo y el paquete TETRAD para aprendizaje en redes bayesianas usando ideas similares. Las mejoras algorítmicas les condujeron a una clara victoria en la copa KDD de 2001 en la competición de minería de datos por un método de aprendizaje de redes bayesianas (Cheng *et al*., 2002). (¡Aquí la tarea específica fue un problema bioinformático con 139.351 características!). Cooper y Herskovits (1992) desarrollaron un enfoque de aprendizaje de la estructura basado en maximizar la verosimilitud. Este enfoque fue mejorado por Heckerman *et al*. (1994). Friedman y Goldszmidt (1996) señalaron la influencia de la representación de las distribuciones locales condicionales en la estructura aprendida.

El problema general del aprendizaje de los modelos probabilísticos con variables ocultas y falta de datos fue abarcado por el algoritmo EM (Dempster *et al*., 1977), el cual fue abstraído a partir de varios métodos existentes, incluido el algoritmo Baum-Welch para aprendizaje HMM (Baum y Petrie, 1966). (El propio Dempster define EM más como un esquema que como un algoritmo, ya que requiere un buen trabajo de cálculo matemático antes de que pueda ser aplicado a una nueva familia de distribuciones.) EM es uno de los algoritmos más ampliamente usados en la ciencia, y McLachlan y Krishnan (1997) dedican un libro entero al algoritmo y sus propiedades. El problema específico de aprendizaje de modelos de mezclas, incluyendo mezclas de gaussianas, se trata en Tittering-ton *et al*. (1985). Dentro de IA, el primer sistema con éxito que usó EM para

modelado de mixturas fue AUTOCLASS (Cheeseman *et al*., 1988; Cheeseman y Stutz, 1996). AUTOCLASS ha sido aplicado a varias tareas de clasificación científica del mundo real, incluyendo el descubrimiento de nuevos tipos de estrellas a partir de datos espectrales (Goebel *et al*., 1989) y nuevas clases de proteínas y de intrones en bases de datos de secuencias de proteínas/ADN (Hunter y States, 1992).

Lauritzen (1995) desarrolló un algoritmo EM para el aprendizaje de redes bayesianas con variables ocultas. También se ha demostrado que las técnicas basadas en el gradiente son efectivas para redes bayesianas al igual que para redes bayesianas dinámicas (Russell *et al*., 1995; Binder *et al*., 1997a). El algoritmo EM estructural fue desarrollado por (Friedman, 1998). La habilidad para aprender la estructura de las redes bayesianas está relacionada con la recuperación de información *causal* a partir de los datos. Esto es, ¿es posible aprender redes bayesianas de forma que la estructura recuperada de la red indique influencias causales reales? Durante muchos años, los estadísticos eludieron esta pregunta, creyendo que los datos observados (en oposición a los datos generados a partir de procesos experimentales) podían producir sólo información correlacional; después de todo, dos variables que aparecen relacionadas podrían, de hecho, verse afectadas por un tercer factor causal desconocido en vez de influir una en otra directamente. Pearl (2000) ha presentado argumentos convincentes de lo contrario, mostrando que existen, de hecho, muchos casos donde la causalidad puede ser determinada y desarrollada por el formalismo **red causal** para expresar las causas y los efectos de las intervenciones, al igual que las probabilidades condicionales ordinarias.

RED CAUSAL

Los modelos de vecinos más cercanos datan de (Fix y Hodges, 1951) y han sido una herramienta estándar en estadística y reconocimiento de patrones desde entonces. Dentro de IA, fueron popularizados por (Stanfill y Waltz, 1986), quienes investigaron métodos para adaptar la distancia métrica a los datos. Hastie y Tibshirani (1996) desarrollaron una forma para localizar la métrica a cada punto del espacio, dependiendo de la distribución de los datos alrededor del punto. Dentro de la comunidad algorítmica, se han estudiado esquemas eficientes de indexamiento para encontrar los vecinos más cercanos (*véase*, por ejemplo, Indyk, 2000). La estimación de la densidad del núcleo, también denominada estimación de la densidad de la **ventana Parzen**, fue investigada inicialmente por Rosenblatt (1956) y Parzen (1962). Desde entonces, se ha desarrollado una enorme cantidad de literatura investigando las propiedades de varios estimadores. Devroye (1987) da una minuciosa introducción.

La literatura sobre redes neuronales es demasiado extensa (aproximadamente 100.000 artículos hasta la fecha) para cubrirla en detalle. Cowan y Sharp (1988b, 1988a) resumen los primeros acontecimientos, comenzando con el trabajo de McCulloch y Pitts (1943). Norbert Wiener, un pionero en la cibernética y en la teoría de control (Wiener, 1948), trabajó con McCulloch y Pitts e influyó en un número de jóvenes investigadores incluyendo a Marvin Minsky, quien pudiera haber sido el primero en desarrollar una red neuronal en *hardware* en 1951 (*véase* Minsky y Papert, 1988, pp. ix-i). Mientras tanto, en Gran Bretaña, W. Ross Ashby (también un pionero de la cibernética; *véase* Ashby, 1940), Alan Turing, Grey Walter y otros, formaron el Club Ratio para «aquellos que tenían las ideas de Wiener antes de que el libro de Wiener apareciera». *Design for a Brain* (Diseño de un cerebro) de Ashby (1948, 1952) propuso la idea de que la inteligencia podría ser creada a través del uso de dispositivos **homeostáticos** que contuvieran ciclos apro-

piados de realimentación para conseguir comportamientos adaptativos estables. Turing (1948) escribió un informe de investigación titulado *Intelligent Machinery* (Maquinaria Inteligente) que comienza con la frase «Propongo investigar si es posible para las máquinas mostrar comportamiento inteligente» y continúa describiendo una arquitectura de redes neuronales recurrentes que denominó «máquinas tipo-B sin organización» y un enfoque para entrenarlas. Desafortunadamente, el informe no se publicó hasta 1969, y fue completamente ignorado hasta hace poco.

Frank Rosenblatt (1957) inventó el «perceptrón» moderno y demostró el teorema de la convergencia del perceptrón (1960), aunque ha sido más mencionado por su trabajo puramente matemático fuera del contexto de las redes neuronales (Agmon, 1954; Motzkin y Schoenberg, 1954). También se hicieron los primeros trabajos sobre redes multicapa, incluyendo los **perceptrones Gamba** (Gamba *et al*., 1961) y **madalines** (Widrow, 1962). *Learning Machines* (Nilsson, 1965) cubre muchos de estos primeros trabajos y otros. La desaparición de los primeros esfuerzos en investigación del perceptrón se aceleró a causa el libro *Perceptrons* (Minsky y Papert, 1969), que lamenta su falta de rigor matemático. El libro señalaba que los perceptrones de una única capa podían representar sólo conceptos linealmente separables y apuntaba la falta de algoritmos de aprendizaje efectivos para redes multicapa.

Se puede considerar que los artículos en (Hinton y Anderson, 1981), basados en la conferencia de San Diego de 1979, marcan el renacimiento del conexionismo. Los dos volúmenes de antologías «PDP» (*Parallel Distributed Processing*) (Rumelhart *et al*., 1986a) y un artículo corto en *Nature* (Rumelhart *et al*., 1986b) atrajeron una gran cantidad de atención; en realidad, el número de artículos sobre «redes neuronales» se ha multiplicado por un factor de 200 entre 1980-84 y 1990-94. El análisis de redes neuronales usando la teoría física de los s*pin-glasses* magnéticos (Amit *et al*., 1985) unió los lazos entre los mecanismos estadísticos y la teoría de redes neuronales, proporcionando no sólo útiles perspicacias matemáticas sino también *respetabilidad*. La técnica de la propagación hacia atrás fue inventada relativamente pronto (Bryson y Ho, 1969) pero se redescubrió varias veces (Werbos, 1974; Parker, 1985).

Las máquinas de vectores soporte se originaron en la década de los 90 (Cortes y Vapnik, 1995) y ahora son sujeto de una literatura que está creciendo rápidamente, incluyendo libros de texto como Cristianini y Shawe-Taylor (2000). Se han hecho muy populares y efectivas para tareas de categorización de textos (Joachims, 2001), investigación bioinformática (Brown *et al*., 2000), y procesamiento del lenguaje natural, como el reconocimiento de dígitos escritos a mano de DeCoste y Scholkopf (2002). Una técnica relacionada que usa también el «truco del núcleo» para representar implícitamente un espacio de características exponencial es el perceptrón con votos (Collins y Duffy, 2002).

La interpretación probabilística de las redes neuronales tiene varias fuentes, incluyendo Baum y Wilczek (1988) y Bridle (1990). Jordan (1995) discute el papel de la función sigmoide. El aprendizaje de los parámetros bayesianos para redes neuronales fue propuesto por MacKay (1992) y fue explorado en más detalle por Neal (1996). La capacidad de las redes neuronales para representar funciones ha sido investigada por Cybenko (1988, 1989), quien mostró que dos capas ocultas son suficientes para representar cualquier función y una única capa es suficiente para representar cualquier función *continua*. El método del «óptimo daño en el cerebro» para eliminar conexiones

inútiles es de LeCun *et al*. (1989), y Siesma y Dow (1988) muestran cómo eliminar unidades inútiles. El algoritmo *tiling* para aumentar estructuras grandes es de Mézard y Nadal (1989). LeCun *et al*. (1995) resumen un número de algoritmos para reconocimiento de dígitos escritos a mano. Desde entonces, Belongie *et al*. (2002) propusieron mejoras en las tasas de error, para encaje de formas y DeCoste y Scholkopf (2002) para vectores soporte virtuales.

Los investigadores de la teoría computacional del aprendizaje han investigado sobre la complejidad del aprendizaje de las redes neuronales. Los primeros resultados computacionales fueron obtenidos por Judd (1990), quien mostró que el problema general de encontrar un conjunto de pesos consistentes con un conjunto de ejemplos es NP-completo, incluso bajo asunciones muy restrictivas. Baum y Haussler (1989) obtuvieron algunos de los primeros resultados sobre la complejidad de la muestra, éstos mostraron que el número de ejemplos requeridos para un aprendizaje efectivo crece aproximadamente $W \log W$, donde W es el número de pesos[16]. Desde entonces, se ha desarrollado una teoría más sofisticada (Anthony y Bartlett, 1999), incluyendo los importantes resultados de que la capacidad de representación de la red depende de la *magnitud* de los pesos al igual que de su número.

FUNCIÓN DE BASE RADIAL

El tipo de red neuronal más popular que no hemos cubierto es la **función de base radial**, o red RBF (*Radial Basis Function*). Una función de base radial combina una colección ponderada de núcleos (normalmente gaussianos, por supuesto) para la aproximación de funciones. La red RBF puede entrenarse en dos fases: primero, se usa un enfoque de agrupamiento no supervisado para entrenar los parámetros de las gaussianas (las medias y las varianzas) como en el Apartado 20.3. En la segunda fase, se determinan los pesos relativos de las gaussianas. Este es un sistema de ecuaciones lineales, que sabemos cómo resolver directamente. Por ello, ambas fases del entrenamiento de la RBF tienen un beneficio: la primera fase es no supervisada, y por ello no requiere datos de entrenamiento etiquetados, y la segunda fase, aunque es supervisada, es eficiente. *Véase* Bishop (1995) para más detalles.

REDES DE HOPFIELD

Las **redes recurrentes**, en las que las unidades están unidas en ciclos, fueron mencionadas en el capítulo, pero no se exploraron en detalle. Las **redes de Hopfield** (Hopfield, 1982) son probablemente la clase de redes recurrentes mejor comprendidas. Usan conexiones *bidireccionales* con pesos *simétricos* (es decir, $W_{i,j} = W_{j,i}$), todas las unidades son a la vez unidades de entrada y unidades de salida, la función de activación g es la función signo, y los niveles de activación sólo pueden ser +1. Una red de Hopfield

MEMORIA ASOCIATIVA

funciona como una **memoria asociativa**: después de que la red se entrena con un conjunto de ejemplos, un nuevo estímulo causará el establecimiento del patrón de activación correspondiente al ejemplo del conjunto de entrenamiento *más parecido* al nuevo estímulo. Por ejemplo, si el conjunto de entrenamiento consiste en un conjunto de fotografías, y un nuevo estímulo es una pequeña parte de una de las fotografías, entonces los niveles de activación de la red reproducirán la fotografía de la cual la parte fue tomada. Nótese que las fotografías originales no se guardan de forma separada en la red; cada peso es una codificación parcial de todas las fotografías. Uno de los resultados teó-

[16] Esto confirma aproximadamente la «regla del tío Bernie». Esta regla lleva el nombre de Bernie Widrow, quien recomendó usar aproximadamente diez veces tantos ejemplos como pesos.

ricos más importantes es que las redes de Hopfield pueden almacenar fidedignamente hasta 0,138N ejemplos de entrenamiento, donde N es el número de unidades de la red.

MÁQUINA BOLTZMANN

Las **máquinas Boltzmann** (Hinton y Sejnowski, 1983, 1986) también usan pesos simétricos, pero incluyen unidades ocultas. Además, usan una función de activación *estocástica*, tal que la probabilidad de que la salida sea 1 es una función de la entrada total ponderada. Por ello, las máquinas Boltzmann experimentan transiciones de estados que recuerdan la búsqueda del recocido simulado (*véase* Capítulo 4) para la configuración que mejor aproxima al conjunto de entrenamiento. Resulta que las máquinas Boltzmann están muy relacionadas con un caso especial de redes bayesianas evaluadas con un algoritmo de simulación estocástica. (*Véase* Apartado 14.5.)

La primera aplicación de las ideas base de las máquinas núcleo fueron de Aizerman *et al*. (1964), pero el desarrollo completo de la teoría, encabezado por las máquinas de vectores soporte, es debido a Vladimir Vapnik y sus colegas (Boser *et al*., 1992; Vapnik, 1998). Cristianini y Shawe-Taylor (2000) y Scholkopf y Smola (2002) proporcionan introducciones rigurosas; una exposición familiar aparece en el artículo *AI Magazine* de Cristianini y Schölkopf (2002).

El material de este capítulo aúna trabajos de los campos de estadística, reconocimiento de patrones y redes neuronales, así que la historia ha sido contada muchas veces de muchas formas. DeGroot (1970), Berger (1985) y Gelman *et al*. (1995) son buenos libros de estadística bayesiana. Hastie *et al*. (2001) proporcionan una introducción excelente a los métodos de aprendizaje estadísticos. Para la clasificación de patrones, el libro clásico durante muchos años ha sido Duda y Hart (1973), ahora actualizado (Duda *et al*., 2001). Para redes neuronales, Bishop (1995) y Ripley (1996) son los libros líderes. El campo de la neurociencia computacional está cubierto por Dayan y Abbott (2001). La conferencia más importante sobre redes neuronales y temas relacionados es la conferencia anual NIPS (*Neural Information Processing Conference*), cuyas actas se publican como series de *Advances in Neural Information Processing Systems*. También aparecen artículos sobre aprendizaje en redes bayesianas en la conferencia *Uncertainty in AI and Machine Learning* y en varias conferencias estadísticas. Algunas revistas específicas sobre redes neuronales incluyen a *Neural Computation*, *Neural Networks* y *IEEE Transactions on Neural Networks*.

EJERCICIOS

20.1 Los datos usados para la Figura 20.1 pueden verse como si hubieran sido generados por h_5. Para cada una de las otras cuatro hipótesis, genere un conjunto de datos de longitud 100 y dibuje los grafos correspondientes a $P(h_i|d_1, ..., d_m)$ y $P(D_{m+1}= lima|d_1, ..., d_m)$. Comente sus resultados.

20.2 Repita el Ejercicio 20.1, esta vez dibujando los valores de $P(D_{m+1} = lima|h_{\text{MAP}})$ y $P(D_{m+1} = lima|h_{\text{ML}})$.

20.3 Suponga que las utilidades de Ann para caramelos de cereza y lima son c_A y ℓ_A, mientras que las utilidades de Bob son c_B y ℓ_B. (Pero una vez que Ann ha desenvuelto un caramelo, Bob no lo querrá comprar.) Presumiblemente, si a Bob le gustan los caramelos de lima más que a Ann, sería prudente para Ann vender su bolsa de caramelos una vez que

ella esté suficientemente segura de su contenido de caramelos de lima. Por otro lado, si Ann desenvuelve demasiados caramelos en el proceso, la bolsa tendrá menos. Discuta el problema de determinar el momento óptimo en el que vender la bolsa. Determine la utilidad esperada del procedimiento óptimo, dada la distribución *a priori* del Apartado 20.1.

20.4 Dos estadísticos van al médico y a ambos se les da el mismo pronóstico: Un 40 por ciento de posibilidad de que el problema sea la enfermedad mortal A, y un 60 por ciento de posibilidad de que sea la enfermedad fatal B. Afortunadamente, existen medicamentos anti-A y anti-B que no son caros, con el 100 por cien de efectividad y libres de efectos secundarios. Los estadísticos tienen la oportunidad de tomar un medicamento, ambos, o ninguno. ¿Qué hará el primer estadista (un aférrimo bayesiano)? ¿Y el segundo estadístico, que siempre usa la hipótesis de máxima verosimilitud?

El médico ha hecho algunas investigaciones y ha descubierto que la enfermedad B realmente se da en dos versiones, dextro-B y levo-B, que son igualmente probables e igualmente tratables por el medicamento anti-B. Ahora hay tres hipótesis, ¿qué harán los dos estadísticos?

20.5 Explique cómo aplicar el método potenciado del Capítulo 18 al aprendizaje de Bayes simple. Compruebe el funcionamiento del algoritmo resultante en el problema de aprendizaje del restaurante.

20.6 Considere m puntos de los datos (x_j, y_j), donde los y_j se generan a partir de los x_j de acuerdo al modelo lineal gaussiano de la Ecuación (20.5). Encuentre los valores de θ_1 y θ_2, y σ que maximizan el logaritmo condicional de la verosimilitud de los datos.

20.7 Considere el modelo *OR ruidoso* para la fiebre descrito en el Apartado 14.3. Explique cómo aplicar el aprendizaje de la máxima verosimilitud para ajustar los parámetros del modelo a un conjunto de datos completos. (Consejo: use la regla de cadena para derivaciones parciales.)

20.8 Este ejercicio analiza las propiedades de la distribución Beta definida en la Ecuación (20.6).

FUNCIÓN GAMMA

a) Integrando en el intervalo [0, 1], muestre que la constante de normalización para la distribución beta$[a, b]$ viene dada por $\alpha = \Gamma(a + b)/\Gamma(a)\,\Gamma(b)$ donde $\Gamma(x)$ es la **función Gamma**, definida por $\Gamma(x + 1) = x \cdot \Gamma(x)$ y $\Gamma(1) = 1$. (Para x enteros, $\Gamma(x + 1) = x!$).

b) Muestre que la media es $a/(a + b)$.

c) Encuentre la(s) moda(s) (los valores más probables de θ).

d) Describa la distribución beta$[\epsilon, \epsilon]$ para ϵ pequeños. ¿Qué ocurre mientras que se actualiza la distribución?

20.9 Considere una red bayesiana arbitraria, un conjunto completo de datos para la red, y la verosimilitud para el conjunto de datos de acuerdo con la red. Proporcione una prueba simple de que la verosimilitud de los datos no puede decrecer si añadimos un nuevo enlace en la red y recalculamos los valores de los parámetros de máxima verosimilitud.

20.10 Considere la aplicación de EM para aprender los parámetros de la red de la Figura 20.10(a), dados los parámetros reales de la Ecuación (20.7).

a) Explique por qué el algoritmo EM no funcionaría si hubiese exactamente dos atributos en el modelo en vez de tres.

b) Muestre los cálculos para la primera iteración de EM empezando a partir de la Ecuación (20.8).

c) ¿Qué ocurre si empezamos con todos los parámetros inicializados al mismo valor p? (Consejo: será útil investigar esto empíricamente antes de obtener el resultado general.)

d) Escriba una expresión para el logaritmo de la máxima verosimilitud para los datos tabulados de los caramelos del Apartado 20.3 en términos de los parámetros, calcule las derivadas parciales respecto a cada parámetro, e investigue la naturaleza de los puntos fijos en el apartado (c).

20.11 Construya a mano una red neuronal que calcule la función XOR de dos entradas. Asegúrese de especificar qué clase de unidades está usando.

20.12 Construya una máquina de vectores soporte que calcule la función XOR. Será conveniente usar valores de 1 y −1 en vez de 1 y 0 para las entradas y para las salidas. Así un ejemplo será de la forma ([−1, 1], 1) o ([−1, −1], −1). Es típico hacer corresponder una entrada **x** con un espacio de cinco dimensiones, las dos dimensiones originales x_1 y x_2, y las tres combinaciones x_1^2, x_2^2 y $x_1\, x_2$. Pero para este ejercicio consideraremos sólo las dos dimensiones x_1 y $x_1\, x_2$. Dibuje los cuatro puntos de entrada en el espacio, y el separador de margen máximo. ¿Cuál es el margen? Ahora dibuje la línea separadora en el espacio original de entrada Euclídeo.

20.13 Un perceptrón simple no puede representar la función XOR (o, en líneas generales, la función paridad de sus entradas). Describa qué ocurre con los pesos, a medida que llegan los ejemplos de la función paridad, en un perceptrón de cuatro entradas con función escalón, comenzando con todos los pesos inicializados a 0,1.

20.14 Recordando del Capítulo 18 que hay funciones boolenas distintas de n entradas. ¿Cuántas de éstas se pueden representar mediante un perceptrón umbral?

20.15 Considere el siguiente conjunto de ejemplos, cada uno con seis entradas y una salida objetivo:

I_1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
I_2	0	0	0	1	1	0	0	1	1	0	1	0	1	1
I_3	1	1	1	0	1	0	0	1	1	0	0	0	1	1
I_4	0	1	0	0	1	0	0	1	0	1	1	1	0	1
I_5	0	0	1	1	0	1	1	0	1	1	0	0	1	0
I_6	0	0	0	1	0	1	0	1	1	0	1	1	1	0
T	1	1	1	1	1	1	0	1	0	0	0	0	0	0

a) Ejecute la regla de aprendizaje del perceptrón en estos datos y muestre los pesos finales.

b) Ejecute la regla de aprendizaje del árbol de decisión, y muestre el árbol de decisión resultante.

c) Comente los resultados.

20.16 Comenzando en la Ecuación (20.13), muestre que $\partial L / \partial W_j = Err \times a_j$.

20.17 Suponga que tiene una red neuronal con funciones de activación lineal. Es decir, para cada unidad, la salida es la suma ponderada de las entradas, un número constante c de veces.

a) Asuma que la red tiene una capa oculta. Para una asignación dada de los pesos **W**, escriba las ecuaciones para el valor de las unidades en la capa de salida como una función de **W** y de la capa de entrada **I**, sin ninguna mención explícita a la salida de las capas ocultas. Muestre que existe una red sin unidades ocultas que calcula la misma función.

b) Repita los cálculos del apartado (a), esta vez para una red con cualquier número de capas ocultas. ¿Qué conclusiones puede obtener sobre las funciones de activación lineal?

20.18 Implemente una estructura de datos para redes neuronales multicapa con alimentación hacia delante, recordando proporcionar la información necesaria tanto para la evaluación hacia delante como para la propagación hacia atrás. Usando esta estructura de datos, escriba una función SALIDA-RED-NEURONAL que dado un ejemplo y una red calcule los valores apropiados de la salida.

20.19 Suponga que un conjunto de entrenamiento contiene sólo un único ejemplo, repetido 100 veces. En 80 de los 100 casos, el único valor de salida es 1; en los otros 20, es 0. ¿Qué predecirá una red con propagación hacia atrás para este ejemplo?, asumiendo que ha sido entrenada y ha alcanzado un óptimo global. (Consejo: para encontrar un óptimo global, diferencie la función del error e iguale el resultado a cero.)

20.20 La red de la Figura 20.24 tiene cuatro nodos ocultos. Este número se escogió de forma arbitraria. Haga experimentos de forma sistemática para medir las curvas de aprendizaje para redes con distinto número de nodos ocultos. ¿Cuál es el número óptimo? ¿Sería posible usar el método de validación cruzada para encontrar con antelación la mejor red?

20.21 Considere el problema de separar N puntos de los datos en ejemplos positivos y negativos usando un separador lineal. Claramente, esto se puede hacer siempre para $N = 2$ puntos con una línea de dimensión $d = 1$, a pesar de cómo estén los puntos etiquetados o dónde estén localizados (a menos que los puntos estén en el mismo lugar).

a) Demuestre que siempre se puede hacer para $N = 3$ puntos sobre un plano de dimensión $d = 2$, a menos que sean colineales.

b) Demuestre que no siempre se puede hacer para $N = 4$ puntos sobre un plano de dimensión $d = 2$.

c) Demuestre que siempre se puede hacer para $N = 4$ puntos sobre un plano de dimensión $d = 3$, a menos que sean coplanarios.

d) Demuestre que no siempre se puede hacer para $N = 5$ puntos sobre un plano de dimensión $d = 3$.

e) El estudiante ambicioso quizá quiera probar que N puntos sobre posiciones generales, pero no $N + 1$, son linealmente separables en un espacio de $N - 1$ dimensiones. A partir de esto se llega a que la **dimensión VC** (que vimos en el Capítulo 18) de espacios lineales en dimensión $N - 1$ es N.

21 Aprendizaje por refuerzo

En este capítulo se examinará cómo un agente puede aprender a partir del éxito y el fracaso, mediante recompensas y castigos.

21.1 Introducción

Los Capítulos 18 y 20 cubren métodos de aprendizaje que aprenden funciones y modelos probabilísticos a partir de ejemplos. En este capítulo, estudiaremos cómo los agentes pueden aprender *qué hacer*, particularmente cuando no hay un profesor que diga al agente qué acción tomar en cada circunstancia.

Por ejemplo, sabemos que un agente puede aprender a jugar al ajedrez con aprendizaje supervisado, proporcionándole ejemplos de situaciones de juego con los mejores movimientos para dichas situaciones. Pero si no hay un profesor que proporcione ejemplos, ¿qué puede hacer el agente? Intentando movimientos aleatorios, el agente puede eventualmente construir un modelo de predicción de su entorno: cómo estará el tablero después de que haga un movimiento e incluso cómo es probable que el oponente responda en una situación dada. El problema es el siguiente: *sin cierta realimentación de lo que es bueno y de lo que es malo, el agente no tendrá razones para decidir qué movimiento hacer*. El agente necesita saber que algo bueno ha ocurrido cuando gana y que algo malo ha ocurrido cuando pierde. Esta clase de realimentación se denomina **recompensa**, o **refuerzo**. En juegos como el ajedrez, el refuerzo se recibe sólo al final del juego. En otros entornos, las recompensas vienen más frecuentemente. En el ping-pong, cada punto que sube al marcador puede considerarse como una recompensa; cuando se aprende a gatear, cualquier movimiento hacia delante es un éxito. Nuestra herramienta para agentes considera la recompensa como *parte* de la entrada que se percibe, pero el agente debe ser «*cableado*» para que reconozca dicha parte como una recompensa en vez de simplemente como otra entrada sensorial. Por ello, los animales parecen estar ca-

bleados para reconocer el dolor o el hambre como recompensas negativas y el placer y la comida se toman como recompensas positivas. Los psicólogos han estudiado cuidadosamente el refuerzo con animales desde hace 60 años.

En el Capítulo 17 se introdujeron las recompensas, que servían para definir políticas óptimas en **procesos de decisión de Markov** (MDPs). Una política óptima es una política que maximiza la recompensa total esperada. La tarea del **aprendizaje por refuerzo** es usar recompensas observadas para aprender una política óptima (o aproximadamente óptima) para el entorno. Mientras que en el Capítulo 17 el agente tiene un modelo completo del entorno y conoce la función de recompensa, aquí no asumimos ningún conocimiento a priori. Imagine jugar a un nuevo juego cuyas reglas desconoce; después de cientos de movimientos más o menos, su oponente dice, «Perdiste». En resumen, esto es aprendizaje por refuerzo.

En muchos dominios complejos, el aprendizaje por refuerzo es el único camino posible para entrenar un programa para que funcione adecuadamente. Por ejemplo, en teoría de juegos, es muy difícil para un humano proporcionar evaluaciones certeras y consistentes de grandes números de posiciones, que serían necesarios para entrenar una función de evaluación directamente desde los ejemplos. En vez de ello, se le puede decir al programa cuándo ha ganado o ha perdido, y puede usar esta información para aprender una función de evaluación que proporcione estimaciones certeras y razonables de la probabilidad de ganar a partir de una situación dada. De forma similar, es extremadamente difícil programar a un agente para que pilote un helicóptero; sin embargo dando recompensas negativas apropiadas por estrellarse, tambalearse, o desviarse de la ruta establecida, un agente puede aprender a volar por sí mismo.

Se considera que el aprendizaje por refuerzo abarca toda la IA: un agente está ubicado en un entorno y debe aprender a comportarse con éxito en él. Para que el capítulo sea manejable, nos concentraremos en marcos sencillos y diseños de agentes sencillos. Para la mayor parte, asumiremos un entorno completamente observable, por ello por cada percepción se facilita el estado actual. Por otro lado, asumiremos que el agente no conoce cómo funciona el entorno o qué acción llevar a cabo, y permitiremos salidas de acciones probabilísticas. Consideraremos tres de los diseños de agentes introducidos por primera vez en el Capítulo 2:

- Un **agente basado en la utilidad** aprende una función de utilidad de los estados y la usa para seleccionar las acciones que maximizan el resultado esperado de la utilidad.
- Un agente de **aprendizaje-*Q*** aprende una función **acción-valor**, o función-*Q*, proporcionando la utilidad esperada de tomar una determinada acción en un estado dado.
- Un **agente reactivo** aprende una política que establece una correspondencia directa entre los estados y las acciones.

APRENDIZAJE-*Q*

ACCIÓN-VALOR

Un agente basado en la utilidad también tiene un modelo del entorno para tomar decisiones, ya que debe conocer los estados a los que sus acciones le llevarán. Por ejemplo, para hacer uso de una función de evaluación del *backgammon*, un programa *backgammon* debe saber cuáles son los movimientos legales *y cómo afectan en la posición del tablero*. Sólo de esta forma puede aplicar la función de utilidad a los estados resultan-

tes. Un agente de aprendizaje-Q, por otro lado, puede comparar los valores de sus posibilidades disponibles sin la necesidad de saber sus resultados, así que no necesita un modelo del entorno. Por otro lado, ya que no sabe dónde le conducen sus acciones, los agentes de aprendizaje-Q no pueden mirar hacia delante; esto puede restringir seriamente su habilidad para aprender, como veremos.

APRENDIZAJE PASIVO

En el Apartado 21.2 empezaremos con **aprendizaje pasivo**, donde la política del agente está fijada y la tarea es aprender las utilidades de los estados (o parejas estado-acción); esto también puede suponer el aprendizaje de un modelo del entorno. El Apartado 21.3 cubre el **aprendizaje activo**, donde el agente debe aprender también qué hacer. El principal aspecto es la **exploración**: un agente debe tener tantas experiencias de su entorno como le sea posible para aprender cómo comportarse en él. El Apartado 21.4 discute cómo un agente puede usar aprendizaje inductivo para aprender más rápido de sus experiencias. El Apartado 21.5 cubre métodos de aprendizaje de representaciones directas de políticas en agentes reactivos. Es esencial para este capítulo entender los procesos de decisión de Markov (Capítulo 17).

APRENDIZAJE ACTIVO

EXPLORACIÓN

21.2 Aprendizaje por refuerzo pasivo

Para que las cosas sean sencillas, comenzamos con el caso de un agente de aprendizaje pasivo que utiliza una representación basada en estados en un entorno completamente observable. En aprendizaje pasivo, la política del agente π está fijada: en el estado s, siempre ejecuta la acción $\pi(s)$. Su meta es sencillamente aprender la bondad de la política, es decir, aprender la función de utilidad $U^{\pi}(s)$. Usaremos nuestro ejemplo del mundo 4 $\times$ 3, introducido en el Capítulo 17. La Figura 21.1 muestra una política para este mundo y sus utilidades correspondientes. De forma clara, la tarea del aprendizaje pasivo es

Figura 21.1 (a) Una política π para el mundo 4×3; esta política es óptima con recompensas de $R(s) = -0{,}04$ en los estados no terminales y sin descuentos. (b) Las utilidades de los estados en el mundo 4×3, dada la política π.

similar a la tarea de la **evaluación de la política**, parte del algoritmo de **iteración de la política** descrito en el Apartado 17.3. La principal diferencia es que el agente de aprendizaje pasivo no conoce el **modelo de transiciones** $T(s, a, s')$, que especifica la probabilidad de alcanzar el estado s' a partir del estado s después de realizar la acción a; ni la **función recompensa** $R(s)$, que especifica la recompensa de cada estado.

PRUEBA

El agente ejecuta un conjunto de **pruebas** en el entorno usando su política π. En cada prueba, el agente comienza en el estado (1, 1) y experimenta una secuencia de transiciones de estados hasta que alcanza uno de los estados terminales, (4, 2) o (4, 3). Sus percepciones proporcionan tanto el estado actual como la recompensa que se recibe en ese estado. Las pruebas típicas tienen el siguiente aspecto:

$$(1, 1)_{-0,04} \rightarrow (1, 2)_{-0,04} \rightarrow (1, 3)_{-0,04} \rightarrow (1, 2)_{-0,04} \rightarrow (1, 3)_{-0,04} \rightarrow (2, 3)_{-0,04} \rightarrow (3, 3)_{-0,04} \rightarrow (4, 3)_{+1}$$
$$(1, 1)_{-0,04} \rightarrow (1, 2)_{-0,04} \rightarrow (1, 3)_{-0,04} \rightarrow (2, 3)_{-0,04} \rightarrow (3, 3)_{-0,04} \rightarrow (3, 2)_{-0,04} \rightarrow (3, 3)_{-0,04} \rightarrow (4, 3)_{+1}$$
$$(1, 1)_{-0,04} \rightarrow (2, 1)_{-0,04} \rightarrow (3, 1)_{-0,04} \rightarrow (3, 2)_{-0,04} \rightarrow (4, 2)_{-1}$$

Nótese que cada estado percibido se subcodifica con la recompensa recibida. El objetivo es usar la información sobre las recompensas para aprender la utilidad esperada $U^{\pi}(s)$ asociada con cada estado no terminal. La utilidad se define como la suma esperada de recompensas (descontadas) obtenidas si se sigue la política π. Como en la Ecuación (17.3) en el Apartado 17.2, esto se escribe como

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right] \tag{21.1}$$

Incluiremos un **factor de descuento** γ en todas nuestras ecuaciones, pero para el mundo 4×3 pondremos $\gamma = 1$.

Estimación directa de la utilidad

ESTIMACIÓN DIRECTA DE LA UTILIDAD

TEORÍA DE CONTROL ADAPTATIVO

Widrow y Hoff (1960) inventaron un método sencillo para la **estimación directa de la utilidad** a finales de la década de los 50 en el área de la **teoría de control adaptativo**. La idea es que la utilidad de un estado es la recompensa total esperada del estado en adelante, y cada prueba proporciona un *ejemplo* del valor para cada estado visitado. Por ejemplo, la primera prueba de las tres dadas anteriormente, proporciona una muestra de recompensa total de 0,72 para el estado (1, 1), dos muestras de 0,66 y 0,84 para (1, 2), dos muestras de 0,80 y 0,88 para (1, 3), etc. Así, al final de cada secuencia, el algoritmo calcula la recompensa observada al llegar a cada estado y actualiza la utilidad estimada para dicho estado, manteniendo una media para cada estado en una tabla. En el límite de infinitas pruebas, la media de la muestra convergerá al valor real esperado en la Ecuación (21.1).

Está claro que la estimación directa de la utilidad es sólo un ejemplo de aprendizaje supervisado donde cada ejemplo tiene como entrada el estado y como salida la recompensa observada por llegar. Esto quiere decir que hemos reducido el aprendizaje por refuerzo a un problema estándar de aprendizaje inductivo, como se discutió en el Capítulo 18. El Apartado 21.4 discute el uso de tipos de representaciones más potentes para las funciones de utilidad, como redes de neuronas. Las técnicas de aprendizaje de dichas representaciones se pueden aplicar directamente a los datos observados.

La estimación directa de la utilidad tiene éxito reduciendo el problema del aprendizaje por refuerzo a un problema de aprendizaje inductivo, sobre el que se conoce más. Desafortunadamente, pierde una fuente muy importante de información, a saber, ¡el hecho de que las utilidades no son independientes! *La utilidad de cada estado es igual a su propia recompensa más la utilidad esperada de sus estados sucesores*. Es decir, los valores de la utilidad obedecen la ecuación de Bellman para una política fija (*véase* también la Ecuación (17.10)):

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^{\pi}(s') \tag{21.2}$$

Ignorando la conexión entre los estados, la estimación directa de la utilidad pierde oportunidades para el aprendizaje. Por ejemplo, la segunda de las tres pruebas dadas anteriormente alcanza el estado (3, 2), que no ha sido previamente visitado. La siguiente transición alcanza (3, 3), del que se conoce a partir de la primera prueba que tiene una alta utilidad. La ecuación de Bellman sugiere inmediatamente que es también probable que (3, 2) tenga una alta utilidad, ya que conduce a (3, 3), pero la estimación directa de la utilidad no aprende nada hasta que se finaliza la prueba. Más ampliamente, podemos ver la estimación directa de la utilidad como una búsqueda en un espacio de hipótesis para U que es más grande de lo que necesita ser, ya que incluye muchas funciones que violan la ecuación de Bellman. Por esta razón, el algoritmo a menudo converge muy lentamente.

Programación dinámica adaptativa

PROGRAMACIÓN DINÁMICA ADAPTATIVA

Para aprovechar las restricciones entre los estados, un agente debe aprender cómo estos están conectados. Un agente de **programación dinámica adaptativa** o **ADP** (*Adaptative Dynamic Programming*) trabaja aprendiendo el modelo de transiciones del entorno a medida que las recorre, y resolviendo el correspondiente proceso de decisión de Markov mediante un método de programación dinámica. Para un agente de aprendizaje pasivo, esto significa introducir el modelo de transiciones aprendido $T(s, \pi(s), s')$ y la recompensa observada $R(s)$ en las ecuaciones de Bellman (21.2), para calcular las utilidades de los estados. Como remarcamos en nuestra discusión sobre la iteración de la política del Capítulo 17, estas ecuaciones son lineales (no suponen maximización) así que se pueden resolver usando cualquier paquete de álgebra lineal. De forma alternativa, podemos adoptar el enfoque de la **iteración de políticas modificado** (*véase* Apartado 17.4), usando un proceso simplificado de iteración del valor para actualizar la utilidad estimada después de cada cambio en el modelo aprendido. Ya que el modelo normalmente cambia sólo un poco con cada observación, el proceso de iteración del valor puede usar la utilidad estimada previamente como valor inicial y debería converger más rápidamente.

El proceso de aprendizaje del modelo en sí mismo es sencillo, ya que el entorno es completamente observable. Esto significa que tenemos una tarea de aprendizaje supervisado donde la entrada es un par estado-acción y la salida es el estado resultante. En el caso más simple, podemos representar el modelo de transición como una tabla de probabilidades. Manteniendo la frecuencia con la que ocurre cada acción de salida y estimando la probabilidad de transición $T(s, a, s')$ a partir de la frecuencia con la que se

alcanza s' cuando se ejecuta a en s[1]. Por ejemplo, en las tres trazas dadas en la página 873, *Derecha* se ejecuta tres veces en (1, 3) y dos de cada tres veces el estado resultante es (2, 3), así se estima que $T((1, 3), \textit{Derecha}, (2, 3))$ es 2/3.

El programa de agente completo para un agente pasivo ADP se muestra en la Figura 21.2. En la Figura 21.3 se muestra su funcionamiento en el mundo 3×4. En términos de la rapidez con que mejora su estimación del valor, el agente ADP hace lo mejor posible, sujeto a su habilidad de aprender el modelo de transiciones. En este sentido, proporciona un estándar con el que medir otros algoritmos de aprendizaje por refuerzo. Sin embargo, esto es en cierto modo intratable para grandes espacios de estados. En el backgammon, por ejemplo, conllevaría resolver aproximadamente 10^{50} ecuaciones con 10^{50} variables desconocidas.

Aprendizaje de diferencia temporal

Es posible tener (casi) lo mejor de ambos mundos; es decir, uno puede aproximar las ecuaciones de restricciones mostradas anteriormente sin resolverlas para todos los posibles estados. *La clave es usar las transiciones observadas para ajustar los valores de los estados observados para que sean acordes con las ecuaciones de restricciones*. Considere, por ejemplo, la transición desde (1, 3) a (2, 3) en la segunda prueba de la página 873. Suponga que, como resultado de la primera prueba, las utilidades estima-

```
función AGENTE-PASIVO-ADP (percepción) devuelve una acción
  entradas: percepción, indica el estado actual s' y la señal de recompensa r'
  estática: π, una política fija
            mdp, un MDP con modelo T, recompensas R, descuento γ
            U, una tabla de utilidades, inicialmente vacía
            N_sa, una tabla de frecuencias para pares estado-acción, inicialmente a cero
            N_sas', una tabla de frecuencias para tripletas estado-acción-estado, inicialmente a cero
            s, a, el estado y la acción previa, inicialmente a nulo (null)

  si s' es nuevo entonces hacer U[s'] ← r' ; R[s'] ← r'
  si s no es nulo (null) entonces hacer
     incrementar N_sa[s, a] y N_sas'[s, a, s']
     para cada t tal que N_sas'[s, a, t] no sea cero hacer
        T[s, a, t] ← N_sas'[s, a, t] / N_sa[s, a]
  U ← DETERMINACIÓN-VALOR(π, U, mdp)
  si TERMINAL?[s'] entonces s, a ← nulo (null) si no s, a ← s', π [s']
  devolver a
```

Figura 21.2 Un agente de aprendizaje por refuerzo pasivo basado en programación dinámica adaptativa. Para simplificar el código, hemos asumido que cada percepción puede dividirse en un estado percibido y una señal de recompensa.

[1] Esta es la estimación de máxima verosimilitud, como se discutió en el Capítulo 20. Una actualización Bayesiana con una distribución a priori Dirichlet funcionaría mejor.

Figura 21.3 Las curvas de aprendizaje pasivo ADP para el mundo 4×3, dada la política óptima mostrada en la Figura 21.1. (a) La utilidad estimada para un subconjunto de estados seleccionados, como una función del número de pruebas. Nótense los grandes cambios que ocurren alrededor de la prueba 78, esta es la primera vez que el agente cae en el estado terminal -1 en (4, 2). (b) La raíz del error cuadrado medio en la estimación de $U(1, 1)$, promediada sobre 20 ejecuciones de 100 pruebas cada una.

das son $U^\pi(1, 3) = 0{,}84$ y $U^\pi(2, 3) = 0{,}92$. Ahora, si esta transición ocurriese todo el tiempo, esperaríamos que las utilidades obedecieran

$$U^\pi(1, 3) = -0{,}04 + U^\pi(2, 3)$$

así que $U^\pi(1, 3)$ sería 0,88. Así, su actual estimación de 0,84 podría ser un poco baja y debería incrementarse. De forma más general, cuando ocurre una transición desde un estado s a otro estado s', aplicamos la siguiente actualización a $U^\pi(s)$:

$$U^\pi(\mathrm{s}) \leftarrow U^\pi(\mathrm{s}) + \alpha(R(s) + \gamma U^\pi(\mathrm{s}') - U^\pi(s)) \tag{21.3}$$

Aquí, α es el parámetro de la **tasa de aprendizaje**. Ya que esta regla actualizada usa la diferencia de utilidades entre estados sucesivos; a menudo se denomina ecuación de **diferencia temporal** o **TD**.

DIFERENCIA TEMPORAL

La idea básica de todos los métodos de diferencia temporal es, primero definir las condiciones que se cumplen localmente cuando la estimación de la utilidad es correcta, y entonces, escribir y actualizar la ecuación que mueve la estimación hacia la ecuación ideal de «equilibrio». En el caso del aprendizaje pasivo, el equilibrio viene dado por la Ecuación (21.2). La Ecuación (21.3) hace que el agente alcance el equilibrio dado por la Ecuación (21.2), pero existe cierta sutileza en todo esto. Primero, nótese que la actualización sólo implica al sucesor observado s', mientras que las condiciones reales de equilibrio implican a todos los posibles estados siguientes. Se podría pensar que esto causa un gran cambio inapropiado en $U^\pi(s)$ cuando ocurre una transición muy poco común; pero, de hecho, ya que las transiciones poco comunes ocurren raramente, el *valor medio* de $U^\pi(s)$ convergerá al valor correcto. Además, si cambiamos α de un parámetro fijo a una función que decrece a medida que incrementa el número de veces que un estado

ha sido visitado, $U(s)$ convergerá al valor correcto[2]. Esto nos proporciona el programa de agente mostrado en la Figura 21.4. La Figura 21.5 ilustra el funcionamiento del agente pasivo TD en el mundo 4×3. No aprende tan rápido como el agente ADP y muestra variabilidad más alta, pero es más sencillo y requiere menos computación por cada observación. Nótese que *TD no necesita un modelo para llevar a cabo sus actualizaciones*. El entorno proporciona las conexiones entre los estados vecinos en forma de transiciones observadas.

función AGENTE-PASIVO-TD (percepción) **devuelve** una acción
  entradas: percepción, una percepción indica el estado actual s' y la señal de recompensa r'
  estática: π, una política fijada
    U, una tabla de utilidades, inicialmente vacía
    N_s, una tabla de frecuencias por estados, inicialmente a cero
    s, a, r, el estado, la acción y la recompensa previa, inicialmente a nulo (null)

  si s' es nuevo **entonces** $U[s'] \leftarrow r'$
  si s no es nulo (null) **entonces hacer**
    incrementar $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$
  si TERMINAL$[s']$ **entonces** $s, a, r \leftarrow$ nulo (null) **si no** $s, a, r \leftarrow s', \pi\,[s'], r'$
  devolver a

Figura 21.4 Un agente de aprendizaje por refuerzo pasivo que aprende estimaciones de la utilidad usando diferencias temporales.

Figura 21.5 Las curvas de aprendizaje TD para el mundo 4×3. (a) La utilidad estimada para un subconjunto seleccionado de estados, como una función del número de pruebas. (b) La raíz del error cuadrado medio en la estimación de $U(1, 1)$, promediada sobre 20 ejecuciones de 500 pruebas cada una. Sólo se muestran las primeras 100 pruebas para permitir la comparación con la Figura 21.3.

[2] Técnicamente, se requiere que $\sum_{n=1}^{\infty} \alpha(n) = \infty$ y $\sum_{n=1}^{\infty} \alpha^2(n) < \infty$. El decrecimiento de $\alpha(n) = 1 / n$ satisface estas condiciones. En la Figura 21.5 hemos usado $\alpha(n) = 60/(59 + n)$.

El enfoque ADP y el enfoque TD están realmente muy relacionados. Ambos intentan hacer ajustes locales a la utilidad estimada para hacer que cada estado «esté de acuerdo» con sus sucesores. Una diferencia es que TD ajusta un estado para que esté de acuerdo con su sucesor *observado* (Ecuación (21.3)), mientras que ADP ajusta el estado para que esté de acuerdo con *todos* los sucesores que pudieran ocurrir, ponderados por sus probabilidades (Ecuación (21.2)). Esta diferencia desaparece cuando los efectos de los ajustes TD se promedian para un gran número de transiciones, ya que la frecuencia de cada sucesor en el conjunto de transiciones es aproximadamente proporcional a su probabilidad. Una diferencia más importante es que mientras TD hace un único ajuste por cada transición observada, ADP hace tantos como necesite para restaurar la consistencia entre la utilidad estimada U y el modelo del entorno T. Aunque la transición observada hace sólo un cambio local en T, sus efectos deberían ser propagados a través de U. Por ello, TD se puede ver como una tosca, pero eficiente, primera aproximación a ADP.

Cada ajuste que hace ADP se puede ver, desde el punto de vista de TD, como un resultado de una «pseudo-experiencia» generada a través de la simulación del modelo actual del entorno. Es posible extender el enfoque de TD para usar un modelo del entorno para generar varias pseudo-experiencias, transiciones que el agente TD puede imaginar que *pudieran* ocurrir, proporcionándolas al modelo actual. Para cada transición observada, el agente TD puede generar un gran número de transiciones imaginarias. De esta forma, la utilidad estimada resultante se aproximará más y más a la de ADP; por supuesto, con el precio de un incremento del tiempo de computación.

En una línea similar, se pueden generar versiones más eficientes de ADP aproximando directamente los algoritmos para iteración de valores o iteración de políticas. Recuérdese que la iteración completa de los valores puede ser intratable cuando el número de estados es grande. Muchos de los pasos de ajuste, sin embargo, son extremadamente pequeños. Un enfoque posible para generar respuestas razonablemente buenas de forma rápida es limitar al número de ajustes que se hacen después de cada transición observada. También se puede usar una heurística para ordenar los posibles ajustes y así llevar a cabo sólo los más significativos. Según la heurística del **barrido con prioridad** es preferible hacer ajustes a estados cuyos sucesores probables han experimentado un *gran número* de ajustes en su propia estimación de utilidad. Usando heurísticas como estas, los algoritmos ADP aproximados pueden aprender casi tan rápido como el ADP completo, en términos del número de secuencias de entrenamiento, pero pueden ser varios órdenes de magnitud más eficientes en términos de computación. (*Véase* Ejercicio 21.3.) Esto les permite manejar espacios de estados que son demasiado grandes para el ADP completo. Los algoritmos ADP aproximados tienen una ventaja adicional: en los primeros pasos del aprendizaje de un nuevo entorno, el modelo del entorno T a menudo estará lejos del correcto, así que hay un pequeño problema en el cálculo de una función de utilidad exacta. Un algoritmo aproximado puede usar un tamaño de ajuste mínimo que decrezca a medida que el modelo del entorno es más preciso. Esto elimina las iteraciones de valor muy largas que pueden ocurrir al principio del aprendizaje debido a los grandes cambios en el modelo.

BARRIDO CON PRIORIDAD

21.3 Aprendizaje por refuerzo activo

Un agente de aprendizaje pasivo tiene una política fija que determina su comportamiento. Un agente activo debe decidir qué acciones tomar. Empecemos con un agente de programación dinámica adaptativa y consideremos cómo debe ser modificado para manejar este nuevo grado de libertad.

Primero, el agente necesitará aprender un modelo completo con salidas probabilísticas para todas las acciones, en vez de aprender sólo el modelo para la política fija. El mecanismo de aprendizaje sencillo que usa el AGENTE-PASIVO-ADP hace esto bien. Luego, necesitamos tener en cuenta el hecho de que el agente tiene que realizar la elección entre las acciones. Las utilidades que necesita aprender son las definidas por la política *óptima*; que obedecen la ecuación de Bellman dada en el Apartado 17.2, que repetimos aquí:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s') \tag{21.4}$$

Estas ecuaciones se pueden resolver para obtener la función de utilidad U usando los algoritmos de iteración de valor o de iteración de política del Capítulo 17. Finalmente hay que determinar qué hacer en cada paso. Habiendo obtenido una función de utilidad U que es óptima para el modelo que se está aprendiendo, el agente puede extraer una acción óptima a partir de «un paso de mirar hacia delante» para maximizar la utilidad esperada; de forma alternativa, si se usa iteración de la política, la política óptima ya está disponible, así que simplemente debería ejecutar la acción que recomienda la política óptima. ¿O no?

Exploración

La Figura 21.6 muestra los resultados de una secuencia de pruebas para un agente ADP que sigue las recomendaciones de una política óptima para el modelo aprendido en cada paso. ¡El agente *no* aprende las verdaderas utilidades o la política óptima verdadera! Lo que ocurre en lugar de esto es que, en la prueba 39, encuentra una política que alcanza la recompensa +1 a través de la ruta más corta por (2, 1), (3, 1), (3, 2) y (3, 3). (*Véase* la Figura 21.6.) Después de experimentos con variaciones mínimas, a partir de la prueba 276 en adelante se queda parado en esa política, sin aprender las utilidades de los otros estados y sin encontrar la ruta óptima a través de (1, 2), (1, 3) y (2, 3). A este agente se le denomina **agente voraz**. Los experimentos repetidos muestran que el agente voraz *rara vez* converge a la política óptima para este entorno y algunas veces converge a políticas realmente horrorosas.

AGENTE VORAZ

¿Cómo puede ser que elegir la acción óptima lleve a resultados subóptimos? La respuesta es que el modelo aprendido no es el mismo que el entorno real; por lo tanto lo que es óptimo en el modelo aprendido puede ser subóptimo en el entorno real. Desafortunadamente, el agente no sabe cuál es el entorno real, así que no puede calcular la acción óptima para el mismo. Entonces, ¿qué se debe hacer?

Lo que el agente voraz ha pasado por alto es que las acciones además de proporcionar recompensas de acuerdo con el modelo aprendido actual, también contribuyen al aprendizaje del modelo verdadero, influyendo en las percepciones que se reciben. Me-

Figura 21.6 Funcionamiento de un agente voraz que ejecuta la acción recomendada por la política óptima del modelo aprendido. (a) Error RMS en la utilidad estimada promediada sobre los nueve cuadrados no terminales. (b) La política subóptima a la que el agente voraz converge en esta particular secuencia de pruebas.

EXPLOTACIÓN

EXPLORACIÓN

jorando el modelo, el agente recibirá mejores recompensas en el futuro[3]. Por ello, un agente debe tener un compromiso entre la **explotación** para maximizar su recompensa (según refleja su estimación actual de la utilidad) y la **exploración** para maximizar su buen comportamiento a largo plazo. Los riesgos de la explotación pura están estancados. La exploración pura para mejorar el conocimiento de uno mismo no tiene sentido si uno nunca pone dicho conocimiento en práctica. En el mundo real, constantemente se tiene que decidir entre continuar en una confortable existencia o entrar en lo desconocido con la esperanza de descubrir una vida nueva y mejor. Con un buen entendimiento, se necesita menos exploración.

PROBLEMAS DEL BANDIDO

¿Podemos ser un poco más precisos que esto? ¿Existe una política de exploración óptima? Resulta que esta pregunta ha sido estudiada en profundidad en el subcampo de la teoría de la decisión estadística que conduce a los denominados **problemas del bandido**. (*Véase* al lado.)

GLIE

Aunque los problemas del bandido son extremadamente difíciles de resolver con exactitud para obtener un método de exploración *óptima*, sin embargo es posible que surja un esquema *razonable* que nos conducirá eventualmente a un comportamiento óptimo por parte del agente. Técnicamente, cualquier esquema necesita ser voraz en el límite de infinitas exploraciones, o **GLIE**. Un esquema GLIE debe intentar cada acción en cada estado un número ilimitado de veces para evitar tener una probabilidad finita de que una acción óptima se pierda por unas series malas de resultados atípicos. Un agente ADP que use dicho esquema aprenderá eventualmente el modelo del entorno real. Un esquema GLIE también debe convertirse eventualmente en voraz, así las acciones del agente se convertirán en óptimas respecto al modelo aprendido (y por ello el verdadero).

[3] Nótese la directa analogía con la teoría del valor de la información del Capítulo 16.

EXPLORACIÓN Y BANDIDOS

En Las Vegas, un *bandido con un solo brazo* es una máquina tragaperras. Un jugador puede insertar una moneda, tirar de la palanca, y recoger las ganancias (si hay). Un **bandido con *n*-brazos** tiene *n* palancas. El jugador debe elegir con qué palanca jugar para cada moneda sucesiva, ¿la que mayor beneficios haya dado, o quizá con la que no haya intentado?

El problema del bandido con *n*-brazos es un modelo formal para problemas reales en muchas áreas de importancia vital, tales como decidir el presupuesto anual para la investigación en IA y su desarrollo. Cada brazo corresponde a una acción (tal como colocar 20 millones de dólares para el desarrollo de nuevos libros de IA), y el resultado de tirar del brazo corresponde a los beneficios obtenidos de tomar la acción (inmensos). La exploración, si es exploración de un nuevo campo de investigación o exploración de un nuevo complejo de tiendas, es arriesgada, es cara, y tiene desenlaces inciertos; por otro lado, no explorar nada significa que uno nunca descubrirá *ningunas* acciones que merezcan la pena.

Para formular un problema del bandido con propiedad, se debe definir exactamente qué se entiende por comportamiento óptimo. La mayoría de las definiciones en la literatura asumen que el propósito es maximizar la recompensa total esperada en el tiempo de vida del agente. Estas definiciones requieren que el valor esperado sea tomado de los posibles mundos en los que el agente pudiera estar, al igual que de los posibles resultados de cada secuencia de acción en cualquier mundo dado. Aquí, un «mundo» se define por el modelo de transiciones $T(s, a, s')$. Por ello, para poder actuar de forma óptima, el agente necesita una distribución a priori de los posibles modelos. Los problemas de optimización resultantes son normalmente intratables.

En algunos casos (por ejemplo, cuando la ganancia de cada máquina es independiente y se utilizan descuentos en las recompensas) es posible calcular el **índice Gittins** para cada máquina tragaperras (Gittins, 1989). El índice es una función del número de veces que se ha jugado con la máquina tragaperras y de la ganancia que ha proporcionado. El índice para cada máquina indica en qué medida merece la pena invertir más, basándose en una combinación de la respuesta esperada y el valor esperado de la información. Elegir la máquina con el valor del índice más alto proporciona una política de exploración óptima. Desafortunadamente, no ha habido forma de extender los índices de Gittins a problemas de decisión secuenciales.

Se puede usar la teoría de los bandidos con *n*-brazos para argumentar la sensatez de la estrategia de selección en los algoritmos genéticos. (*Véase* Capítulo 4.) Si se considera cada brazo en el problema del bandido con *n*-brazos como una posible cadena de genes, y la inversión de una moneda en un brazo como la reproducción de dichos genes, los algoritmos genéticos colocan las monedas de forma óptima, proporcionando un conjunto apropiado de asunciones de independencia.

Existen varios esquemas GLIE; uno de los más sencillos consiste en permitir que el agente elija una acción aleatoria una fracción $1/t$ del tiempo y que siga la política avara en el resto de los casos. Aunque esto converge eventualmente a una política óptima, puede ser extremadamente lento. Un enfoque más sensato proporcionaría algún peso a las acciones que el agente no ha intentado muy a menudo, mientras que tendería a evitar las acciones que tienen una baja utilidad. Esto puede ser implementado alterando la ecuación de restricción (21.4) para que asigne una mayor estimación de la utilidad a pares estado-acción inexplorados. Esencialmente, esto alcanza una distribución a priori optimista sobre los posibles entornos y hace que el agente se comporte inicialmente como si tuviera recompensas maravillosas dispersas por todos los lugares. Usaremos $U^+(s)$ para

denotar la estimación optimista de la utilidad (es decir, la «recompensa por llegar» esperada) del estado s, y sea $N(a, s)$ el número de veces que la acción a se ha intentado en s. Suponga que estamos usando iteración del valor en un agente de aprendizaje ADP; entonces necesitamos reescribir la ecuación de actualización (es decir, la Ecuación (17.6)) para incorporar la estimación optimista. La siguiente ecuación hace esto:

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(a,s)\right) \tag{21.5}$$

FUNCIÓN EXPLORACIÓN

A $f(u, n)$ se le denomina **función exploración**. Determina el compromiso entre la voracidad (preferencia por valores altos de u) y la curiosidad (preferencia por valores bajos de n: acciones que no se intentan a menudo). La función $f(u, n)$ debería ser incrementada en u y decrementada en n. Obviamente, existen muchas funciones posibles que cumplen estas condiciones. Una definición particular sencilla es

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{en otro caso} \end{cases}$$

donde R^+ es una estimación optimista de la mejor recompensa posible que se puede obtener en cualquier estado y N_e es un parámetro fijo. Esto tendrá el efecto de hacer que el agente intente cada par acción-estado al menos N_e veces.

El hecho de que aparezca U^+ en vez de U en el lado derecho de la Ecuación (21.5) es muy importante. Mientras que la exploración continúa, los estados y las acciones cercanas al estado de comienzo deberían intentarse un gran número de veces. Si usáramos U, la estimación de utilidad más pesimista, entonces rápidamente el agente no estaría dispuesto a explorar más. El uso de U^+ supone que los beneficios de la exploración se propaguen hacia atrás desde los bordes de las regiones sin explorar, así las acciones que llevan *hacia* regiones inexploradas tienen más peso, en vez de las que no resultan familiares. El efecto de esta política de exploración se puede ver de forma clara en la Figura 21.7, que muestra una convergencia rápida a un rendimiento óptimo, a diferencia del enfoque vo-

Figura 21.7 Funcionamiento de un agente ADP con exploración, usando $R^+ = 2$ y $N_e = 5$. (a) Estimación de la utilidad para estados seleccionados sobre el tiempo. (b) El error RMS en los valores de la utilidad y la política asociada perdida.

raz. Después de sólo 18 pruebas se encuentra una política cercana a la óptima. Nótese que la propia estimación de la utilidad no converge tan rápidamente. Esto es porque el agente deja de explorar las partes del espacio de estados que no tienen recompensa, visitándolos sólo «por accidente». Sin embargo, para el agente tiene perfecto sentido no preocuparse por las utilidades exactas de los estados que no son deseables y pueden ser evitados.

Aprendizaje de una Función Acción-Valor

Ahora que tenemos un agente ADP activo, consideremos cómo construir un agente de aprendizaje activo de diferencia temporal. El cambio más obvio del caso pasivo es que el agente no está equipado con una política fija, así que, si aprende una función de utilidad U, necesitará aprender un modelo para ser capaz de elegir una acción basada en U a través de un paso de mirada hacia delante. El problema de la adquisición del modelo para el agente TD es idéntico al del agente ADP. ¿Qué actualización del TD se regula a sí misma? Quizá sorprendentemente, la regla de actualización (21.3) permanezca sin cambios. Esto puede parecer extraño, por la siguiente razón: suponga que el agente realiza un paso que normalmente le lleva a un buen destino, pero debido a un entorno no determinístico el agente acaba en un estado catastrófico. La regla de actualización TD tomará esto tan seriamente como si el resultado hubiese sido el resultado normal de la acción, mientras que uno supondría que, ya que el resultado fue casual, el agente no debería preocuparse demasiado por ello. De hecho, por supuesto, ocurrirá un resultado improbable sólo de forma infrecuente en un gran conjunto de secuencias de entrenamiento; por lo tanto en ejecuciones largas sus efectos serán ponderados de forma proporcional a su probabilidad, como esperaríamos. Otra vez, puede mostrarse que el algoritmo TD convergerá a los mismos valores que ADP cuando el número de secuencias de entrenamiento tiende a infinito.

Existe un método TD alternativo denominado **aprendizaje**-Q que aprende una representación acción-valor en vez de aprender utilidades. Usaremos la notación $Q(a, s)$ para denotar el valor de realizar la acción a en el estado s. Los valores-Q están directamente relacionados con los valores de utilidad de la siguiente forma:

$$U(s) = \max_a Q(a,s) \tag{21.6}$$

LIBRE DE MODELO

Las funciones-Q parecen otra forma de almacenar información de utilidad, pero tienen una propiedad importante: *un agente TD que aprende una función-Q no necesita un modelo ni para el aprendizaje ni para la selección de acciones*. Por esta razón, el aprendizaje-Q se denomina método **libre de modelo** (*model-free*). Como con las utilidades, podemos escribir una ecuación de restricción que se mantenga en el equilibrio cuando los valores-Q sean correctos:

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s') \tag{21.7}$$

Como en el agente de aprendizaje ADP, podemos usar esta ecuación directamente como una ecuación de actualización para un proceso iterativo que calcule los valores-Q exactos, dado un modelo estimado. Esto hace, sin embargo, que se requiera también el

aprendizaje del modelo, ya que la ecuación usa $T(s, a, s')$. El enfoque de diferencia temporal, por otro lado, no requiere ningún modelo. La ecuación de actualización para el aprendizaje-Q TD es

$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)) \tag{21.8}$$

que se calcula siempre que se ejecuta una acción a en el estado s que conduce al estado s'.

En la Figura 21.8 se muestra el diseño completo del agente para un agente de aprendizaje-Q exploratorio que usa TD. Nótese que usa exactamente la misma función de exploración f que la usada por el agente ADP exploratorio, de ahí la necesidad de mantener las estadísticas de las acciones tomadas (la tabla N). Si se usa una política de exploración simple (es decir, actuar aleatoriamente en algunos pasos, donde el número de pasos decrece con el tiempo) podemos prescindir de las estadísticas.

El agente de aprendizaje-Q aprende la política óptima para el mundo 4×3, pero lo hace más lentamente que el agente ADP. Esto es porque TD no obliga a que se cumpla la consistencia entre los valores a través del modelo. La comparación plantea una cuestión general: ¿es mejor aprender un modelo y una función de utilidad que aprender una función acción-valor sin modelo? En otras palabras, ¿cuál es la mejor forma de representar la función del agente? Este es un aspecto de básico de la inteligencia artificial. Como empezamos en el Capítulo 1, una de las características claves de la historia de la mayoría de la investigación en IA es su adherencia al enfoque **basado en el conocimiento**. Esto equivale a la asunción de que la mejor forma de representar la función del agente es construir una representación de algunos aspectos del entorno en el cual el agente está ubicado.

Algunos investigadores, tanto dentro como fuera de IA, han reivindicado que la disponibilidad de métodos libres de modelo tales como el aprendizaje-Q significa que el enfoque basado en el conocimiento es innecesario. Sin embargo, esta intuición está poco justificada. Nuestra intuición, sobre lo que merece la pena, dice que cuando el

```
función AGENTE-APRENDIZAJE-Q (percepción) devuelve una acción
  entradas: percepción, una percepción indica el estado actual s' y la señal de recompensa r'
  estática: Q, una tabla de valores de acción indexada por el estado y la acción
            N_sa, una tabla de frecuencias de los pares estado-acción
            s, a, r, el estado, la acción y la recompensa previa, inicialmente nulos

  si s no es nulo entonces hacer
     incrementar N_sa[s, a]
     Q[s, a] ← Q[a, s] + α (N_sa[s, a]) (r + γ max_a' Q[a', s'] − Q[a, s])
  si TERMINAL?[s'] entonces s, a, r ← nulo
  si no s, a, r ← s', argmax_a' f(Q[a', s'], N_sa[a', s'], r'
  devolver a
```

Figura 21.8 Un agente de aprendizaje-Q exploratorio. Es un aprendizaje activo que aprende el valor $Q(a, s)$ de cada acción en cada situación. Usa la misma función de exploración f que el agente ADP exploratorio, pero evita tener que aprender el modelo de transiciones ya que el valor-Q de un estado se puede relacionar directamente con los de sus vecinos.

entorno se convierte en más complejo, las ventajas de un enfoque basado en el conocimiento se manifiestan más. Esto ocurre incluso en juegos como el ajedrez, las damas, y el *backgammon* (*véase* la siguiente sección), donde los esfuerzos para aprender una función de evaluación mediante un modelo han tenido más éxito que los métodos de aprendizaje-Q.

21.4 Generalización en aprendizaje por refuerzo

Hasta ahora, hemos asumido que las funciones de utilidad y las funciones-Q aprendidas por los agentes se representan en forma tabular, con un valor de salida para cada tupla de entrada. Este enfoque funciona razonablemente bien para espacios de estados pequeños, pero el tiempo de convergencia y (para ADP) el tiempo por iteración se incrementa rápidamente a medida que el espacio se hace más grande. Con un control cuidadoso, los métodos ADP aproximados pueden hacer posible el manejo de 10.000 estados o más. Esto es suficiente para entornos bidimensionales como el del laberinto, pero los mundos más realistas se quedan fuera. El ajedrez y el *backgammon* son un pequeño subconjunto del mundo real, sus espacios de estados contienen del orden de 10^{50} a 10^{120} estados. ¡Sería absurdo suponer que se pueden visitar todos estos estados para aprender cómo jugar al juego!

APROXIMACIÓN DE FUNCIONES

Una forma de manejar estos problemas es utilizar **aproximación de funciones**, que consiste en representar la función de forma distinta a una tabla. La representación se ve como una aproximación porque puede no darse el caso de que la función *verdadera* de utilidad o función-Q se pueda representar en la forma elegida. Por ejemplo, en el Capítulo 6 se describió una **función de evaluación** para el ajedrez que se representa como una función lineal ponderada de un conjunto de características (o **funciones base**) $f_1, ..., f_n$:

FUNCIONES BASE

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2 + \cdots + \theta_n f_n(s)$$

Un aprendizaje por refuerzo puede aprender valores para los parámetros $\theta = \theta_1, ..., \theta_n$ de forma que la función de evaluación $\hat{U}_\theta$ aproxime la función de utilidad verdadera. En vez de 10^{120} valores en la tabla, esta aproximación se caracteriza por $n = 20$ parámetros, una compresión *enorme*. Aunque nadie conoce la verdadera función de utilidad para el ajedrez, nadie se cree que se pueda representar exactamente con 20 números. Sin embargo, si la aproximación es lo suficientemente buena, el agente podrá jugar un ajedrez excelente[4].

La aproximación de funciones es práctica para representar funciones para espacios de estados muy grandes, pero éste no es su beneficio principal. La *compresión que se consigue con un aproximador de una función permite al agente de aprendizaje generalizar a partir de estados que ha visitado a estados que no ha visitado*. Esto es, el aspecto

[4] Sabemos que la función de utilidad exacta se puede representar en una página o dos de Lisp, Java o C++. Esto es, se puede representar por un programa que resuelva el juego de forma exacta cada vez que es llamado. Estamos interesados sólo en funciones de aproximación que utilicen una cantidad de cálculo *razonable*. De hecho, podría ser mejor aprender una aproximación de función muy simple, y combinarla con cierta cantidad de búsqueda hacia delante. Los inconvenientes que esto implica todavía no se conocen bien.

más importante de la aproximación de funciones no es que requiera menos espacio, sino que permite hacer generalización inductiva sobre los espacios de entrada. Para darle una idea de la potencia de este efecto: examinando sólo uno de cada 10^{44} de los posibles estados del *backgammon*, es posible aprender una función de utilidad que permite a un programa jugar tan bien como los humanos (Tesauro, 1992).

Por otro lado, por supuesto, está el problema de que pudiera no haber una función en el espacio de hipótesis elegido que aproximara suficientemente bien la función de utilidad verdadera. Como en todo aprendizaje inductivo, hay un compromiso entre el tamaño del espacio de hipótesis y el tiempo que se toma para aprender la función. Un espacio de hipótesis grande incrementa la probabilidad de que se encuentre una buena aproximación, pero también supone que la convergencia se retrase.

Comencemos con el caso más simple, que es una estimación directa de la utilidad. (*Véase* el Apartado 21.2). Con aproximación de funciones, esto es una instancia de **aprendizaje supervisado**. Por ejemplo, suponga que representamos las utilidades para el mundo 4×3 utilizando una función lineal simple. Las características de los cuadrados son exactamente sus coordenadas x e y, tenemos

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y \tag{21.9}$$

Así, si $(\theta_0, \theta_1, \theta_2) = (0, 5, 0, 2, 0, 1)$, entonces $\hat{U}_\theta(1, 1) = 0{,}8$. Dada una colección de pruebas, obtenemos un conjunto de valores de muestra $\hat{U}_\theta(x, y)$ y podemos encontrar el mejor ajuste, en el sentido de minimizar el error cuadrado, utilizando regresión lineal estándar. (*Véase* Capítulo 20).

Para el aprendizaje por refuerzo, tiene más sentido utilizar un algoritmo de aprendizaje *en línea* que actualice los parámetros después de cada prueba. Suponga que ejecutamos una prueba y la recompensa total obtenida empezando en (1, 1) es 0,4. Esto sugiere que $\hat{U}_\theta(1, 1)$, actualmente con valor 0,8, es muy grande y debe ser reducido. ¿Cómo se deben ajustar los parámetros para conseguir esto? Como con el aprendizaje de redes neuronales, escribimos una función de error y calculamos su gradiente respecto a los parámetros. Si la recompensa total observada desde el estado s hacia delante en la j-ésima prueba es $u_j(s)$, el error se define como (la mitad) de la diferencia al cuadrado entre el total predicho y el total actual: $E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$. La tasa de cambio del error respecto a cada parámetro θ_i es $\partial E_j/\partial\theta_i$, por lo tanto, para variar el parámetro en la dirección en que el error se decrementa, queremos

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha(u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \tag{21.10}$$

REGLA DE WIDROW-HOFF

REGLA DELTA

Esta se denomina **regla de Widrow-Hoff**, o **regla delta**, para mínimos cuadrados *en línea*. Para el aproximador lineal de función $\hat{U}_\theta(s)$ en la Ecuación (21.9) obtenemos tres sencillas reglas de actualización:

$$\theta_0 \leftarrow \theta_0 + \alpha(u_j(s) - \hat{U}_\theta(s))$$
$$\theta_1 \leftarrow \theta_1 + \alpha(u_j(s) - \hat{U}_\theta(s))x$$
$$\theta_2 \leftarrow \theta_2 + \alpha(u_j(s) - \hat{U}_\theta(s))y$$

Podemos aplicar estas reglas al ejemplo donde $\hat{U}_\theta$ (1, 1) es 0,8 y u_j(1, 1) es 0,4. θ_0, θ_1 y θ_2 se decrementan todas en 0,4α, lo que reduce el error para (1, 1). Observe que ¡cambiar las θ_is *también modifica los valores de* $\hat{U}_\theta$ *para cada uno de los otros estados*! Esto es lo que queremos decir cuando decimos que la aproximación de función permite a un sistema de aprendizaje por refuerzo generalizar a partir de sus experiencias.

Esperamos que el agente aprenda más rápidamente si utiliza un aproximador de función, dado que el espacio de hipótesis no es muy grande, pero incluye algunas funciones que son, de forma razonable, un buen ajuste de la función de utilidad verdadera. El Ejercicio 21.7 le invita a evaluar el rendimiento de la estimación directa de la utilidad, de ambas formas, utilizando y sin utilizar aproximador de función. La mejora en el mundo 4×3 es notable, pero no dramática, porque es un espacio de estados muy pequeño para comenzar con él. La mejora es mucho mayor en un mundo 10×10 con una recompensa de +1 en (10, 10). Este mundo está bien adecuado para una función de utilidad lineal, porque la función de utilidad verdadera es plana y prácticamente lineal. (*Véase* Ejercicio 21.10). Si ponemos una recompensa de +1 en (5, 5), la función de utilidad verdadera es más parecida a una pirámide, y lamentablemente, el aproximador de función fallará. ¡Sin embargo, no está todo perdido! Recuerde que lo que pasa en la aproximación de funciones lineal es que la función es lineal en los *parámetros;* las características en sí mismas pueden ser funciones arbitrarias no lineales de las variables de estado. Así que podemos incluir un término como $\theta_3 = \sqrt{(x - x_g)^2 + (y - y_g)^2}$ que mida la distancia a la meta.

Estas ideas se pueden aplicar igualmente a sistemas de aprendizaje de diferencia temporal. Todo lo que necesitamos hacer es ajustar los parámetros e intentar reducir la diferencia temporal entre estados sucesivos. Las nuevas versiones de las ecuaciones TD y aprendizaje-Q (21.3 y 21.8) son

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i} \tag{21.11}$$

para utilidades y

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \max_{a'} \hat{Q}_\theta(a', s') - \hat{Q}_\theta(a, s)] \frac{\partial \hat{Q}_\theta(a, s)}{\partial \theta_i} \tag{21.12}$$

para valores-Q. Se puede demostrar que estas reglas de actualización convergen a la aproximación más cercana posible[5] a la función verdadera, cuando la función es *lineal* en los parámetros. Desafortunadamente no se puede decir lo mismo cuando se utilizan funciones *no lineales,* como redes neuronales. Hay algunos casos simples en los que los parámetros pueden irse a infinito, incluso aunque haya buenas soluciones en el espacio de hipótesis. Hay algoritmos más sofisticados que evitan estos problemas, pero actualmente en el aprendizaje por refuerzo con aproximadores generales de funciones sigue siendo un arte delicado.

La aproximación de funciones puede ser también muy útil para aprender un modelo del entorno. Recuerde que aprender un modelo de un entorno *observable* es un pro-

[5] La definición de distancia entre funciones de utilidad es bastante técnica; *véase* Tsitsiklis y Van Roy (1997).

blema de aprendizaje *supervisado*, porque la siguiente percepción proporciona el estado resultado. Se puede utilizar cualquiera de los métodos de aprendizaje supervisado del Capítulo 18, con los ajustes adecuados para tener en cuenta el hecho de que necesitamos predecir una descripción completa del estado en vez de una clasificación booleana o un valor real simple. Por ejemplo, si el estado se define mediante n variables booleanas, necesitaremos aprender n funciones booleanas para predecir todas las variables. Para un entorno *parcialmente observable*, el problema del aprendizaje es mucho más difícil. Si sabemos cuántas variables ocultas hay, y cómo se relacionan de forma causal con otras variables observables, podemos fijar la estructura de una red Bayesiana dinámica y utilizar el algoritmo EM para aprender los parámetros, como describimos en el Capítulo 20. La invención de las variables ocultas y el aprendizaje de la estructura del modelo aún son problemas abiertos.

Ahora volvemos a ejemplos de aplicaciones de gran escala del aprendizaje por refuerzo. Veremos que, en los casos en que se utiliza una función de utilidad (y por lo tanto un modelo), normalmente el modelo se toma como dado. Por ejemplo, en el aprendizaje de una función de evaluación para el *backgammon*, normalmente se asume que son conocidos sus movimientos legales y sus efectos.

Aplicaciones a juegos

La primera aplicación significativa del aprendizaje por refuerzo fue también el primer problema significativo de aprendizaje en general, el programa jugador de ajedrez escrito por Arthur Samuel (1959, 1967). Inicialmente, Samuel utilizó una función lineal ponderada para la evaluación de las posiciones, utilizando hasta 16 términos en un momento dado. Él aplicó una versión de la Ecuación (21.11) para actualizar los pesos. Sin embargo, existen algunas diferencias significativas entre su programa y los métodos actuales. Primero, él actualizaba los pesos utilizando la diferencia entre el estado actual y el valor almacenado generado mediante búsqueda hacia delante completa en el árbol de búsqueda. Esto funciona bien, porque equivale a ver el espacio de estados con diferente granularidad. Una segunda diferencia es que ¡el programa *no* utilizaba ninguna recompensa observada! Esto es, los valores de los estados terminales eran ignorados. Esto significa que es bastante posible que el programa de Samuel no converja, o que converja a una estrategia diseñada para perder en vez de para ganar. Para evitar esta suerte, él insistía en que los pesos para tener ventaja material deben ser siempre positivos. Sorprendentemente, esto era suficiente para dirigir el programa hacia áreas del espacio de pesos correspondientes a un buen jugador de ajedrez.

El sistema TD-Gammon de Gerry Tesauro (1992) ilustra, de forma contundente, el potencial de las técnicas de aprendizaje por refuerzo. En sus primeros trabajos (Tesauro y Sejnowski, 1989), Tesauro intentó aprender una representación de red neuronal de $Q(a, s)$, directamente a partir de ejemplos de movimientos etiquetados por un experto con valores relativos. Este enfoque resultó extremadamente tedioso para el experto. Dio lugar a un programa denominado NEUROGAMMON, que fue fuerte con los estándar de computador, pero no competitivo contra expertos humanos. El proyecto TD-Gammon fue un intento de aprender únicamente a partir del juego propio. La única señal de recompen-

sa se daba al final de cada juego. La función de evaluación se representaba mediante una red neuronal completamente conectada, con una capa oculta que contenía 40 nodos. Simplemente mediante la aplicación repetida de la Ecuación (21.11), TD-Gammon aprendió a jugar considerablemente mejor que Neurogammon, incluso aunque la representación de entrada contenía exactamente la posición del tablero cruda, sin características computadas. Esto supuso alrededor de 200.000 juegos de entrenamiento y dos semanas de tiempo de computación. Aunque puede parecerse a muchos juegos, es sólo una pequeña fracción despreciable del espacio de estados. Cuando se añadieron características precomputadas a la representación de entrada, una red con 80 unidades ocultas fue capaz, después de 300.000 juegos de entrenamiento, de alcanzar un estándar de juego comparable al de los tres mejores jugadores del mundo. Kit Woolsey, un gran jugador y analista dijo: «No cabe duda en mi mente de que su juicio posicional es mucho mejor que el mío.»

Aplicación a control de robots

EL CARRO Y LA PÉRTIGA

PÉNDULO INVERTIDO

En la Figura 21.9 se muestra el esquema del famoso problema de equilibrio del **carro y la pértiga** también conocido como el **péndulo invertido**. El problema consiste en controlar la posición x del carro, de forma que la pértiga se quede prácticamente vertical ($\theta \approx \pi/2$), cuando los límites de la pista en la que está el carro son los que se muestran. En relación con este problema, aparentemente simple, se han publicado más de 2.000 artículos en aprendizaje por refuerzo y teoría de control. El problema del carro y la pértiga difiere de los problemas descritos anteriormente en que las variables de estado x, θ, $\dot{x}$ y $\dot{\theta}$ son continuas. Las acciones normalmente son discretas: tirar a la izquierda o tirar a la derecha, el denominado régimen de **control bang-bang**.

CONTROL BANG-BANG

Figura 21.9 Esquema del problema de equilibrar una larga pértiga que está encima de un carro en movimiento. El carro se puede mover a izquierda y derecha por un controlador que observa x, θ, $\dot{x}$ y $\dot{\theta}$.

El primer trabajo de aprendizaje para este problema fue llevado a cabo por Michie y Chambers (1968). Su algoritmo BOXES fue capaz de equilibrar la pértiga más de una hora después de sólo 30 pruebas. Además, al contrario que muchos sistemas posteriores, BOXES se implementó con un carro y una pértiga reales, no con simulación. El algoritmo primero discretiza en cajas (*boxes*) el espacio de estados de cuatro dimensiones (de ahí el nombre). Ejecuta pruebas hasta que la pértiga cae o el carro llega al final de la pista. Con la acción final en la caja final, se asociaba una recompensa negativa que se propagaba hacia atrás en la secuencia. Se descubrió que la discretización causaba algunos problemas cuando el aparato se inicializaba en una posición diferente de las utilizadas en el entrenamiento, lo que sugiere que la generalización no era perfecta. Se puede obtener una mejora de la generalización y un aprendizaje más rápido utilizando un algoritmo que haga particiones *adaptadas* del espacio de estados de acuerdo con la variación observada de la recompensa. Hoy en día, un ejercicio común es equilibrar un péndulo invertido *triple* (una proeza que está lejos de las capacidades de la mayoría de los humanos).

21.5 Búsqueda de la política

BÚSQUEDA DE LA POLITICA

El enfoque final que consideraremos para problemas de aprendizaje por refuerzo se denomina **búsqueda de la política**. De alguna forma, la búsqueda de la política es el método más sencillo de todos los de este capítulo: la idea es mantener la política dando vueltas tanto tiempo como el rendimiento mejore, y después parar.

Empecemos con las propias políticas. Recuerde que una política π es una función que hace corresponder estados a acciones. Estamos interesados principalmente en representaciones *parametrizadas* de π, que tengan menos parámetros que estados hay en el espacio de estados (exactamente como en la sección anterior). Por ejemplo, podríamos representar π mediante una colección de funciones-Q parametrizadas, una para cada acción, y tomar la acción con mayor valor predicho:

$$\pi(s) = \max_a \hat{Q}_\theta(a, s) \tag{21.13}$$

Cada función-Q podría ser una función lineal de parámetros θ, como en la Ecuación (21.9), o podría ser una función no lineal como una red neuronal. Después, la búsqueda de la política ajustará los parámetros θ para mejorar la política. Observe que si la política se representa mediante funciones-Q, la búsqueda de la política es un proceso que aprende funciones-Q. ¡*Este proceso no es el mismo que el aprendizaje-Q*! En el aprendizaje-Q con aproximación de función, el algoritmo encuentra un valor de θ tal que $\hat{Q}_\theta$ es «cercana» a Q^*, la función-Q óptima. Por otro lado, la búsqueda de la política encuentra el valor de θ que proporciona un mejor rendimiento; los valores encontrados pueden diferir muy substancialmente[6]. Otro ejemplo claro de la diferencia es el caso en que $\pi(s)$

[6] De forma trivial, la función-Q aproximada, definida por $\hat{Q}_\theta(a, s) = Q^*(a, s)/10$ proporciona rendimiento óptimo, incluso aunque no está del todo cercana a Q^*.

se calcula utilizando búsqueda hacia delante de profundidad 10 con una función de utilidad aproximada $\hat{U}_\theta$. El valor de θ que proporciona buen rendimiento es una forma de hacer que $\hat{U}_\theta$ se parezca a la función de utilidad verdadera.

Un problema con las representaciones de la política como las de la Ecuación (21.13) es que la política es una función *discontinua* de los parámetros cuando las acciones son discretas[7]. Esto es, habrá valores de θ tales que un cambio infinitesimal en θ cause que la política cambie de una acción a otra. Esto significa que el valor de la política también puede cambiar de forma discontinua, lo que hace complicada la búsqueda basada en el gradiente. Por esta razón, los métodos de búsqueda de la política normalmente utilizan una representación **estocástica de la política** $\pi_\theta(s, a)$, que especifica la *probabilidad* de seleccionar la acción a en el estado s. Una representación popular es la **función softmax**:

POLÍTICA ESTOCASTICA

FUNCION SOFTMAX

$$\pi_\theta(s, a) = \exp(\hat{Q}_\theta(a, s)) / \sum_{a'} \exp(\hat{Q}_\theta(a', s))$$

Softmax es prácticamente determinística si una acción es mucho mejor que las demás, pero siempre proporciona una función de θ diferenciable; por eso, el valor de la política (que depende de una forma continua de las probabilidades de selección de acciones) es una función de θ diferenciable.

Ahora, veamos los métodos para mejorar la política. Empezamos con el caso más simple: una política determinística en un entorno determinístico. En este caso, evaluar la política es trivial: simplemente la ejecutamos, y observamos la recompensa acumulada; esto nos proporciona el **valor de la política** $\rho(\theta)$. Mejorar la política es un problema estándar de optimización, como se describía en el Capítulo 4. Podemos seguir el vector **gradiente de la política** $\nabla_\theta \rho(\theta)$, dado que $\rho(\theta)$ es diferenciable. De forma alternativa, podemos seguir el **gradiente empírico** mediante escalada, es decir, evaluando el cambio en la política para incrementos pequeños de cada valor de parámetro. Con las advertencias usuales, este proceso convergerá a un óptimo local en el espacio de políticas.

VALOR DE LA POLÍTICA

GRADIENTE DE LA POLITICA

Cuando el entorno (o la política) es estocástico, las cosas son más complicadas. Suponga que estamos intentando hacer escalada, lo que requiere comparar $\rho(\theta)$ y $\rho(\theta + \Delta\theta)$ para algún $\Delta\theta$ pequeño. El problema es que la recompensa total de cada prueba puede variar enormemente, por lo que estimar el valor de la política a partir de un número de pruebas pequeño será muy poco fiable; intentar comparar dos de estas estimaciones será aún menos fiable. Una solución es simplemente ejecutar muchas pruebas, midiendo las varianzas de muestra y utilizarlas para determinar si se han ejecutado suficientes pruebas para obtener una indicación fiable de la dirección de mejora para $\rho(\theta)$. Desafortunadamente, esto es impracticable para muchos problemas reales, donde cada prueba puede ser muy cara, consumir mucho tiempo, y quizás ser peligrosa.

Para el caso de una política estocástica $\pi_\theta(s, a)$ es posible obtener una estimación imparcial del gradiente en θ, $\nabla_\theta \rho(\theta)$, directamente de los resultados de las pruebas ejecutadas en θ. Por simplicidad, obtendremos esta estimación para el caso simple de un entorno no secuencial en el que la recompensa se obtiene inmediatamente después de

[7] Para un espacio de acciones continuo, la política puede ser una función plana de los parámetros.

actuar en el estado inicial s_0. En este caso, el valor de la política es exactamente el valor esperado de la recompensa, y tenemos

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s_0, a)R(a) = \sum_a (\nabla_\theta \pi_\theta(s_0, a))R(a)$$

Ahora, realizamos un simple truco, de forma que este sumatorio pueda ser aproximado mediante muestras generadas a partir de la distribución de probabilidad definida por $\pi_\theta(s_0, a)$. Suponga que tenemos N pruebas y que la acción tomada en la prueba j-ésima es a_j. Entonces

$$\nabla_\theta \rho(\theta) = \sum_a \pi_\theta(s_0, a) \cdot \frac{(\nabla_\theta \pi_\theta(s_0, a))R(a)}{\pi_\theta(s_0, a)} \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s_0, a_j))R(a_j)}{\pi_\theta(s_0, a_j)}$$

Así, el verdadero gradiente del valor de la política se aproxima mediante una suma de términos que incluyen el gradiente de la probabilidad de selección de acción en cada prueba. Para el caso secuencial, esto se generaliza a

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s_0, a_j))R_j(s)}{\pi_\theta(s_0, a_j)}$$

para cada estado s visitado, donde a_j se ejecuta en s en la j-ésima prueba y $R_j(s)$ es la recompensa total recibida del estado s en adelante en la j-ésima prueba. El algoritmo resultante se denomina REINFORCE (Williams, 1992); normalmente es mucho más efectivo que la escalada utilizando muchas pruebas en cada valor de θ. Sin embargo, es todavía más lento de lo necesario.

Considere la siguiente tarea: dados dos programas de blackjack[8], determinar cuál es el mejor. Una forma de hacer esto es que cada uno juegue contra un «*croupier*» estándar un número determinado de manos, y después mida sus victorias respectivas. El problema con esto, como hemos visto, es que las victorias de cada programa fluctúan enormemente dependiendo de si recibe cartas buenas o malas. Una solución obvia es generar un determinado número de manos por adelantado. De esta forma, eliminamos el error que se debe a la diferencia de cartas recibidas. Esta es la idea en la que se basa el algoritmo PEGASUS (Ng y Jordan 2000). El algoritmo es aplicable a dominios para los que está disponible un simulador y se pueden repetir los resultados «aleatorios» de las acciones. El algoritmo funciona generando por adelantado N secuencias de números aleatorios, cada uno de los cuales se puede utilizar para ejecutar una prueba de cualquier política. La búsqueda de la política se lleva a cabo evaluando cada política candidata haciendo uso del *mismo* conjunto de secuencias aleatorias para determinar los resultados de las acciones. Se puede demostrar que el número de secuencias aleatorias que se requieren para asegurar que el valor de *cada* política está bien estimado, depende únicamente de la complejidad del espacio de políticas, y no de la complejidad del dominio subyacente. El algoritmo PEGASUS ha sido utilizado para desarrollar políticas efectivas en varios dominios, incluyendo vuelo autónomo de helicópteros (*véase* Figura 21.10).

[8] También conocido como ventiuna o pontón.

Figura 21.10 Imágenes superpuestas en un período de tiempo de un helicóptero autónomo realizando una peligrosa maniobra circular. El helicóptero está bajo el control de una política desarrollada por el algoritmo de búsqueda de la política PEGASUS. Se desarrolló un modelo simulador observando los efectos de varias manipulaciones de control en helicópteros reales; el algoritmo fue ejecutado en el modelo simulador por la noche. Se desarrollaron una gran variedad de controladores para diferentes maniobras. En todos los casos, el rendimiento obtenido excedió por mucho el de un piloto humano experto utilizando control remoto. (La imagen es cortesía de Andrew Ng.)

21.6 Resumen

Este capítulo ha examinado el problema de aprendizaje por refuerzo: cómo un agente puede hacerse experto en un entorno desconocido, únicamente a partir de sus propias percepciones y recompensas ocasionales. El aprendizaje por refuerzo se puede ver como un microcosmos para el problema completo de la IA, pero se estudia en esquemas simplificados para facilitar el progreso. Los puntos más importantes son:

- El diseño global del agente dicta el tipo de información que debe ser aprendida. Los tres principales diseños que hemos cubierto son diseños basados en el modelo, utilizando un modelo T y una función de utilidad U; un diseño de modelo libre que utiliza una función Q acción-valor; y el diseño reactivo que utiliza una política π.
- Las utilidades se pueden aprender utilizando tres enfoques:
  - — La **estimación directa de la utilidad** utiliza la recompensa total observada por llegar a un estado dado como evidencia directa para aprender su utilidad.
  - — La **programación dinámica adaptativa** (ADP) aprende un modelo y una función de recompensa a partir de las observaciones y después utiliza un valor o política de iteración para obtener las utilidades o una política óptima. ADP hace

un uso óptimo de las restricciones sobre utilidades de estados impuestas mediante la estructura de vecindad del entorno.

— Los métodos de **diferencia temporal** (TD) actualizan las estimaciones de utilidad para que correspondan con las de los estados sucesores. Se puede ver como aproximaciones simples al enfoque ADP, que no requieren modelo para el proceso de aprendizaje. Sin embargo, utilizar un modelo aprendido para generar pseudoexperiencias puede dar como resultado un aprendizaje más rápido.

- Las funciones acción-valor, o funciones-Q, se pueden aprender mediante el enfoque ADP o mediante un enfoque TD. Con TD, el aprendizaje-Q no requiere modelo ni en la fase de aprendizaje ni en la de selección de acciones. Esto simplifica el problema de aprendizaje, pero restringe potencialmente la habilidad de aprender en entornos complejos, porque el agente no puede simular los resultados de los posibles cursos de acción.
- Cuando el agente de aprendizaje es responsable de seleccionar acciones mientras aprende, debe compensar entre el valor estimado de las acciones y el potencial de aprender nueva información útil. Una solución exacta para el problema de exploración es imposible, pero algunas heurísticas simples realizan un trabajo razonable.
- En espacios de estados grandes, los algoritmos de aprendizaje por refuerzo deben utilizar una representación funcional aproximada para generalizar sobre los estados. La señal de diferencia temporal se puede utilizar directamente para actualizar los parámetros en representaciones como redes neuronales.
- Los métodos de **búsqueda de la política** operan directamente sobre la política de representación, intentando mejorarla con base enl rendimiento observado. La varianza del rendimiento en un dominio estocástico es un problema serio; para dominios simulados se puede superar fijando la aleatoriedad de antemano.

Por su potencial para eliminar estrategias de control codificadas a mano, el aprendizaje por refuerzo continúa siendo una de las áreas más activas de investigación en aprendizaje automático. Las aplicaciones en robótica prometen ser particularmente valiosas; requerirán métodos para manejar entornos *continuos*, de *muchas dimensiones* y *parcialmente observables* en los que los comportamientos con éxito pueden consistir en miles o incluso millones de acciones primitivas.

Notas bibliográficas e históricas

Turing (1948, 1950) propuso el enfoque del aprendizaje por refuerzo, aunque no estaba convencido de su efectividad, escribiendo, «el uso de castigos o recompensas puede como mucho ser una parte del proceso de enseñanza». El trabajo de Arthur Samuel (1959), fue probablemente el primero con éxito en la investigación sobre aprendizaje automático. Aunque su trabajo fue informal y tenía algunos defectos, contenía la mayoría de las ideas modernas del aprendizaje por refuerzo. Hacia la misma época, los investigadores en teoría de control adaptativa (Widrow y Hoff, 1960), basándose en el trabajo de Hebb (1949), entrenaron redes simples con la regla delta. (Esta conexión inicial en-

tre las redes neuronales y el aprendizaje por refuerzo ha podido llevar a la persistente idea equivocada de que el segundo es un subcampo del primero.) El trabajo en el carro y la pértiga de Michie y Chambers (1968) también se puede ver como un método de aprendizaje por refuerzo con un aproximador de función. La literatura psicológica en aprendizaje por refuerzo es mucho más antigua; Hilgard y Bower (1975) proporcionan un buen resumen. Las investigaciones sobre el extraño comportamiento de las abejas han proporcionado la evidencia directa de la aplicación del aprendizaje por refuerzo en animales; hay una correlación neural clara de la señal de recompensa, en forma de una gran correspondencia neural a partir de los sensores de entrada de néctar, directamente con la corteza motora (Montague *et al*., 1995). La investigación utilizando grabación de celda única sugiere que el sistema de dopamina en cerebros de primates implementa algo parecido al aprendizaje de la función valor (Schultz *et al*. 1997).

La conexión entre el aprendizaje por refuerzo y los procesos de decisión de Markov fue inicialmente realizada por Werbos (1977), pero el desarrollo de aprendizaje por refuerzo en IA es resultado del trabajo en la Universidad de Massachusetts a comienzos de los años 80 (Barto *et al*., 1981). El artículo de Sutton (1988) proporciona un buen resumen histórico. La Ecuación (21.3) de este capítulo es un caso especial para $\lambda = 0$ del algoritmo general TD(λ) de Sutton. TD(λ) actualiza los valores de todos los estados en una secuencia, llevando a cada transición mediante una cantidad que disminuye como λ^t para estados de t pasos anteriores. TD(1) es idéntico a la regla de Widrow-Hoff o regla delta. Boyan (2002), basándose en el trabajo de Bradtke y Barto (1996) argumenta que TD(λ) y los algoritmos relacionados hacen un uso ineficiente de las experiencias; esencialmente, son algoritmos de regresión *en línea* que convergen mucho más lentamente que la regresión *fuera de línea*. Su algoritmo *en línea* LSTD(λ) proporciona los mismos resultados que la regresión *fuera de línea*.

La combinación del aprendizaje de diferencia temporal con la generación basada en modelos de experiencias simuladas fue propuesta en la arquitectura DYNA de Sutton (Sutton, 1990). La idea de barrido con prioridades fue introducida independientemente por Moore y Atkeson (1993) y Peng y Williams (1993). El aprendizaje-Q fue desarrollado en la tesis doctoral de Watkin (1989).

Los problemas del bandido, que modelizan el problema de exploración de decisiones no secuenciales, fueron analizados en profundidad por Berry y Fristedt (1985). Las estrategias óptimas de exploración para algunos esquemas se pueden obtener utilizando la técnica denominada **índices Gittins** (Gittins, 1989). Barto *et al*. (1995) discute una variedad de métodos de exploración para problemas de decisiones secuenciales. Kearns y Singh (1998) y Brafman y Tennenholtz (2000) describen algoritmos que exploran entornos desconocidos para los que está garantizada la convergencia a políticas cercanas a la óptima en un tiempo exponencial.

La aproximación de funciones en aprendizaje por refuerzo viene del trabajo de Samuel, que utilizó funciones de evaluación lineales y no lineales, y métodos de selección de características para reducir el espacio de las mismas. Los métodos posteriores incluyen el **CMAC** (*Cerebellar Model Articulation Controller*) (Albus, 1975), que esencialmente es una suma de funciones núcleo local que se superponen, y las redes neuronales asociativas de Barto *et al*. (1983). Actualmente, las redes neuronales son la forma más popular de aproximación de funciones. La aplicación que se conoce mejor es TD-Gam-

CMAC

mon (Tesauro, 1992, 1995), que se ha discutido en este capítulo. Un problema significatico que presenta el aprendizaje TD con redes neuronales es que tiende a olvidar las experiencias que han ocurrido hace tiempo, especialmente aquellas en partes del espacio de estados que se evitaron una vez que se ha conseguido competencia. Esto puede dar lugar a fallos catastróficos si reaparecen estas circunstancias. La aproximación de funciones basada en **aprendizaje basado en instancias** puede evitar este problema (Ormoneit y Sen, 2002; Forbes, 2002).

La convergencia de los algoritmos de aprendizaje por refuerzo que utilizan aproximación de funciones es una cuestión extremadamente técnica. Los resultados para aprendizaje TD han sido reforzados progresivamente para el caso de aproximadores lineales de funciones (Sutton, 1988; Dayan, 1992; Tsitsiklis y Van Roy, 1997), pero se han presentado algunos ejemplos de divergencia para funciones no lineales (veáse Tsitsiklis y Van Roy, 1997, para una discusión). Papavassiliou y Russell (1999) describen un nuevo tipo de aprendizaje por refuerzo que converge con cualquier forma de aproximador de función, dado que se puede encontrar una aproximación de mejor ajuste para los datos observados.

Los métodos de búsqueda de política fueron llevados a la fama por parte de Williams (1992), que desarrolló la familia de algoritmos REINFORCE. Los trabajos posteriores de Marbach y Tsitsiklis (1998), Sutton *et al.* (2000) y Baxter y Bartlett (2000) reforzaron y generalizaron los resultados de convergencia para la búsqueda de la política. El algoritmo PEGASUS se debe a Ng y Jordan (2000), aunque en la tesis doctoral de Van Roy (1998) aparecen técnicas similares. Como hemos mencionado en este capítulo, el rendimiento de una política *estocástica* es una función continua de sus parámetros, que se mejora con métodos de búsqueda basados en el gradiente. Este no es el único beneficio: Jaakkola *et al.* (1995) argumentan que realmente, las políticas estocásticas funcionan mejor que las políticas determinísticas en entornos parcialmente observables, si ambas se limitan a actuar con base en la percepción actual. (Una razón es que es menos probable que la política estocástica se atasque debido a algunos obstáculos que no se han visto.) En el Capítulo 17 apuntamos que las políticas óptimas en MDPs parcialmente observables son funciones determinísticas del *espacio de creencias* en vez de funciones de la percepción actual, por lo que aún esperaríamos mejores resultados manteniendo el espacio de creencias haciendo uso de los métodos de **filtrado** del Capítulo 15. Desafortunadamente, el espacio de estados de creencia es de dimensiones altas y continuo, y todavía no se han desarrollado algoritmos efectivos para aprendizaje con refuerzo con estados de creencia.

Los entornos del mundo real presentan una enorme complejidad, en términos del número de acciones primitivas que se requieren para conseguir una recompensa significativa. Por ejemplo, un robot puede realizar cientos de miles de movimientos de piernas antes de meter un gol. La **determinación de la recompensa** es un método común, que se utilizaba originalmente en entrenamiento de animales. Supone proporcionar al agente recompensas adicionales para «hacer progresos». Para el fútbol, éstas se pueden dar por hacer contacto con el balón o por lanzarlo hacia la portería. Estas recompensas pueden acelerar enormemente el aprendizaje, y se pueden incluir fácilmente, pero existe el riesgo de que el agente aprenda a maximizar las pseudo-recompensas en vez de las recompensas reales; por ejemplo, estar cerca del balón y «vibrar» causa muchos contac-

DETERMINACIÓN DE LA RECOMPENSA

tos con el mismo. Ng *et al*. (1999) muestran que el agente seguirá aprendiendo la política óptima dado que la pseuso-recompensa $F(s, a, s')$ satisface $F(s, a, s') = \gamma\Phi(s') = \Phi(s)$, donde Φ es una función arbitraria del estado. Se puede construir Φ de forma que refleje cualquier aspecto deseable del estado, tal como la consecución de submetas o la distancia a un estado meta.

APRENDIZAJE POR REFUERZO JERARQUICO

Los **métodos jerárquicos de aprendizaje por refuerzo**, que intentan resolver problemas en múltiples niveles de abstracción (muy parecidos a los **métodos de planificación HTN** del Capítulo 12), también pueden facilitar la generación de comportamientos complejos. Por ejemplo, «meter un gol» se puede descomponer en «obtener la posesión», «regatear hacia la portería» y «disparar»; y cada una de éstas se puede descomponer en más comportamientos motores de bajo nivel. Los resultados fundamentales en este área se deben a Forestier y Varaiya (1978), que demostraron que los comportamientos de bajo nivel de complejidad arbitraria se pueden tratar como acciones primitivas (aun aquellos que pueden variar en el tiempo) desde el punto de vista de los comportamientos de alto nivel que los invocan. Los enfoques actuales, Parr y Russell (1998), Dietterich (2000), Sutton *et al*. (2000), Andre y Russell (2002), se basan en este resultado para desarrollar métodos que provean a un agente con un **programa parcial** que restringe el comportamiento del agente de forma que tenga una estructura jeráquica. Después, se aplica aprendizaje por refuerzo para aprender el mejor comportamiento consistente con el programa parcial. La combinación de aproximación de funciones, la *determinación* de la recompensa y el aprendizaje por refuerzo jerárquico permite enfrentarse a problemas de gran escala de forma satisfactoria.

PROGRAMA PARCIAL

El resumen de Kaelbling *et al*. (1996) es un buen punto de entrada a la literatura. El texto de Sutton y Barto (1998), dos de los pioneros del campo, se centra en las arquitecturas y algoritmos, mostrando cómo el aprendizaje por refuerzo entrelaza las ideas de aprendizaje, planificación y actuación. El trabajo algo más técnico de Bertsekas y Tsitsiklis (1996) proporciona de forma rigurosa los conocimientos básicos en la teoría de programación dinámica y convergencia estocástica. Los artículos de aprendizaje por refuerzo se publican frecuentemente en *Machine Learning*, *Journal of Machine Learning Research* y en las Conferencias Internacionales de *Machine Learning* y *Neural Information Processing Systems*.

EJERCICIOS

21.1 Implemente un agente de aprendizaje pasivo en un entorno simple como el mundo 4×3. Para el caso de un modelo del entorno inicialmente desconocido, compare el rendimiento del aprendizaje de los algoritmos de estimación directa de la utilidad, TD y ADP. Realice la comparación para la política óptima y para varias políticas aleatorias. ¿Para cuál converge más rápidamente la estimación de la utilidad? ¿Qué pasa cuando se aumenta el tamaño del entorno? (Intente entornos con y sin obstáculos).

21.2 El Capítulo 17 definía una **política adecuada** para un MDP, como aquella para la que está garantizado alcanzar un estado terminal. Muestre que para un agente ADP pasivo es posible aprender un modelo de transición para el que su política π es inade-

cuada (incluso si π es adecuada para el MDP verdadero; con estos modelos, el paso de determinación de valores puede fallar si $\gamma = 1$). Muestre que este problema no puede surgir si la determinación de valores se aplica al modelo aprendido sólo al final de una prueba.

21.3 Modifique el agente ADP pasivo para que utilice un algoritmo ADP aproximado como el discutido en el texto. Haga esto en dos pasos:

a) Implemente una cola de prioridades para ajustes de la estimación de la utilidad. Siempre que se ajusta un estado, todos sus predecesores también se convierten en candidatos para ajuste y deben ser añadidos a la cola. La cola se inicializa con el estado para el que ha tenido lugar la transición más reciente. Permita sólo un número fijo de ajustes.

b) Experimente con varias heurísticas para ordenar la cola de prioridades, examinando su efecto en las tasas de aprendizaje y el tiempo de computación.

21.4 El método de estimación directa de la utilidad del Apartado 21.1 utiliza estados terminales distintos para indicar el final de una prueba. ¿Cómo se podría modificar para entornos con descuento de recompensas y sin estados terminales?

21.5 ¿Cómo se puede utilizar el algoritmo de determinación de valores para calcular la pérdida esperada experimentada por un agente que utiliza un conjunto dado de estimaciones de utilidad U y un modelo estimado M, comparado con un agente que usa los valores correctos?

21.6 Adapte el mundo de la aspiradora (Capítulo 2) para aprendizaje por refuerzo incluyendo recompensas por recoger cada porción de suciedad, por guardarse y por apagarse. Haga el mundo accesible proporcionando las percepciones adecuadas. Ahora experimente con distintos agentes de aprendizaje por refuerzo. ¿Es la aproximación de funciones necesaria para el éxito? ¿Qué tipo de aproximador funciona para esta aplicación?

21.7 Implemente un agente de exploración de aprendizaje por refuerzo que utilice la estimación directa de la utilidad. Construya dos versiones: una con una representación tabular y otra que utilice el aproximador de función de la Ecuación (21.9). Compare su rendimiento en tres entornos.

a) El mundo 4×3 descrito en este capítulo.

b) El mundo 10×10 sin obstáculos y con una recompensa de +1 en (10, 10).

c) El mundo 10×10 sin obstáculos y con una recompensa de +1 en (5, 5).

21.8 Escriba la actualización de parámetros para el aprendizaje TD con

$$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sqrt{(x - x_g)^2 + (y - y_g)^2}$$

21.9 Invente características adecuadas para un mundo de cuadrícula estocástica (generalización del mundo 4×3) que contenga múltiples obstáculos y estados terminales múltiples con recompensas de +1 o −1.

21.10 Calcule la función de utilidad verdadera y la mejor aproximación lineal en x e y (como en la Ecuación (21.9)) para los siguientes entornos:

a) Un mundo 10×10 con un único estado terminal +1 en (10, 10).
b) Como (a), pero añadiendo un estado terminal −1 en (10, 1).
c) Como (b), pero añadiendo obstáculos en 10 cuadrados seleccionados aleatoriamente.
d) Como (b), pero situando una pared en el tramo que va desde (5, 2) a (5, 9).
e) Como (a), pero con el estado terminal en (5, 5).

Las acciones son movimientos determinísticos en las cuatro direcciones. En cada caso, compare los resultados utilizando gráficos tridimensionales. Para cada entorno, proponga características adicionales (además de x e y) que mejorarían la aproximación y muestre los resultados.

21.11 Extienda el entorno estándar de juegos (Capítulo 6) para que incorpore una señal de recompensa. Sitúe dos agentes de aprendizaje por refuerzo en el entorno (por supuesto, pueden compartir el programa de agente) y deje que jueguen uno contra el otro. Aplique la regla de actualización TD generalizada (Ecuación (21.11)) para actualizar la función de evaluación. Podría empezar con una simple función de evaluación lineal ponderada y un juego sencillo como el tic-tac-toe.

21.12 Implemente los algoritmos REINFORCE y PEGASUS y aplíquelos al mundo 4×3, utilizando una familia de políticas de su propia elección. Comente los resultados.

21.13 Investigue la aplicación de las ideas de aprendizaje por refuerzo para modelizar el comportamiento humano y animal.

21.14 ¿Es el aprendizaje por refuerzo un modelo abstracto apropiado para la evolución?¿Qué conexión existe, si existe alguna, entre señales de recompensa cableadas y la capacidad evolutiva?

22 La comunicación

En este capítulo veremos por qué los agentes pueden querer intercambiar mensajes que lleven información y cómo lo pueden hacer.

Anochece en los bosques de la sabana del Parque Nacional de Amboseli a los pies del Kilimanjaro. Un grupo de monos vervet («vervet monkeys») está buscando comida cuando uno de ellos emite un ruidoso aullido. Los otros individuos del grupo lo reconocen como el aviso de la presencia de un leopardo (distinto del corto carraspeo usado para prevenir de las águilas o del chasquido para las serpientes) y trepan por los árboles. El vervet se ha comunicado satisfactoriamente con el grupo.

COMUNICACIÓN

SIGNOS

La **comunicación** es el intercambio intencional de información efectuado mediante la producción y percepción de **signos** pertenecientes a un sistema compartido de signos convencionales. La mayoría de los animales usan signos para representar mensajes importantes: comida aquí, depredador cercano, acercamiento, retirada, compañero. En un mundo observable parcialmente la comunicación puede ayudar a los agentes a prosperar ya que pueden adquirir información que es observada o inferida por los demás.

LENGUAJE

Lo que hace diferente a los humanos de los otros animales es el complejo sistema de mensajes estructurados, que conocemos como **lenguaje**, y que nos permite comunicar la mayor parte de las cosas que sabemos acerca del mundo. Aunque los chimpancés, los delfines y otros mamíferos han dado muestra de manejar vocabularios de cientos de signos y cierta aptitud para encadenarlos, únicamente los humanos pueden comunicar fiablemente un número ilimitado de mensajes cualitativamente diferentes.

Desde luego, hay otros atributos que son exclusivamente humanos: ninguna otra especie viste ropas, crea arte figurativo o ve tres horas diarias la televisión. Pero cuando Turing propuso su prueba (*véase* Apartado 1.1) la basó en el lenguaje, ya que el lenguaje está íntimamente vinculado con el pensamiento. En este capítulo explicaremos cómo trabaja y describe un fragmento de castellano un agente comunicativo.

22.1 La comunicación como acción

ACTO DE HABLA

Una de las acciones que puede realizar un agente es la de producir lenguaje. Esta acción es llamada un **acto de habla**. «Habla» es usado en el mismo sentido que «libertad de palabra», no estrictamente hablado de forma que, el correo electrónico, las señales de humo y el uso del lenguaje de signos son considerados todos ellos como actos de habla. En el idioma inglés no existe una palabra específica para designar a un agente que produce lenguaje sea cual sea el medio, por ello usaremos **hablante**, **oyente** y **declaración** como términos genéricos para referirnos a cualquier modo de comunicación. También emplearemos el término **palabra** para referirnos a cualquier clase de signo comunicativo convencional.

HABLANTE

OYENTE

DECLARACIÓN

PALABRA

¿Por qué querrá un agente molestarse en realizar un acto de habla cuando puede hacer una acción «normal»? Hemos visto en el Capítulo 12 que los agentes en un entorno multiagente pueden servirse de la comunicación para lograr planes comunes. Por ejemplo, un grupo de agentes que está explorando simultáneamente el mundo de *wumpus*[1] obtiene más ventajas (colectiva e individualmente) si es capaz de:

- **Preguntar** a otros agentes acerca de aspectos particulares del mundo. Esto se suela hacer con cuestiones como: *¿Has olfateado al wumpus en alguna parte?*
- **Informarse** entre sí acerca del mundo. Esto se hace con declaraciones representativas: *Hay una brisa aquí, en 3 4*. La respuesta a una pregunta es otra clase de acción informativa.
- **Solicitar** a otros agentes la realización de acciones: *Por favor, ayúdame a llevar el oro*. Algunas veces se considera más correcto usar un **acto de habla indirecto** (una solicitud en forma de declaración o pregunta): *Podría utilizar alguna ayuda para cargar esto*. Un agente con autoridad puede dar órdenes (*Alpha ve a la derecha; Bravo y Charlie id a la izquierda*) y un agente con poder puede amenazar (*Dame el oro, pues si no…*). A esta clase de actos de habla se les llama **directivas**.
- **Aceptación** de peticiones: *De acuerdo*.
- **Prometer** o comprometerse a un plan: *Yo dispararé al wumpus; mientras tú coge el oro*.

ACTO DE HABLA INDIRECTO

Todos los actos de habla afectan al mundo al hacer vibrar las moléculas del aire (o el efecto equivalente en cualquier otro medio) y cambiando el estado mental y finalmente las acciones futuras de otros agentes. Algunos tipos de actos de habla transfieren información al oyente, asumiendo que la toma de decisiones del oyente se verá afectada por dicha información. Otros tienen la intención directa de hacer que el oyente haga alguna acción. Otra clase de actos de habla, las **declaraciones**, surgen para tener un efecto más directo sobre el mundo, como en *Os declaro marido y mujer* o *tres, estás fuera*. Desde luego, el efecto es conseguido creando o confirmando una compleja red de estados mentales entre los agentes involucrados: estar casado o estar fuera del juego son estados caracterizados principalmente por convenciones más que por propiedades «físicas» del mundo.

DECLARACIONES

[1] *Nota del traductor: Wumpus* es el monstruo central de una familia de videojuegos de los años 70 llamado «Hunt The Wumpus».

La tarea de los agentes que se comunican es decidir *cuándo* se le solicita un acto de habla de alguna clase y *qué* acto de habla, de entre todas las posibilidades, es el correcto. El problema de entender los actos de habla es similar a otros problemas de **comprensión**, tales como interpretar imágenes o diagnosticar enfermedades. Se nos da un conjunto de entradas ambiguas y a partir de ellas debemos retroceder para decidir qué estado del mundo pudo haber creado esas entradas. Sin embargo, debido a que el habla es una acción planificada, su comprensión también implica el reconocimiento de planes.

COMPRENSIÓN

Fundamentos del lenguaje

LENGUAJE FORMAL

CADENAS

SÍMBOLOS TERMINALES

LENGUAJES NATURALES

Un **lenguaje formal** se define como un (posiblemente infinito) conjunto de **cadenas**. Cada cadena es a su vez una concatenación de **símbolos terminales**, algunas veces llamados palabras. Por ejemplo, en el lenguaje de la lógica de primer orden, $\wedge$ y P son símbolos terminales y «$P \wedge Q$» es una cadena típica. La cadena «$P\ Q \wedge$» no pertenece al lenguaje. Los lenguajes formales, tales como la lógica de primer orden o Java, tienen definiciones matemáticas estrictas. Esto contrasta con los **lenguajes naturales**, tales como chino, danés, inglés y castellano, que no tienen definiciones estrictas y son usados por una comunidad de hablantes. En este capítulo intentaremos tratar los lenguajes naturales como si ellos fueran lenguajes formales, aunque reconocemos que el emparejamiento no será perfecto.

GRAMÁTICA

Una **gramática** es un conjunto finito de reglas que especifican un lenguaje. Los lenguajes formales siempre tienen una gramática oficial, especificada en manuales o libros. Los lenguajes naturales tienen gramáticas no oficiales, pero los lingüistas se afanan en descubrir propiedades del lenguaje por medio de un proceso de investigación científica y entonces codificar sus descubrimientos en una gramática. Hasta la fecha, ningún lingüista lo ha conseguido completamente. Obsérvese que los lingüistas son científicos, que intentan definir cómo *es* un lenguaje. Hay también gramáticos prescriptivos, los cuales intentan dictar cómo *debe ser* un lenguaje. Ellos crean reglas tales como «No dividir infinitivos» que son algunas veces escritas en guías de estilo, pero que tienen poca relevancia en el uso actual del lenguaje.

Tanto los lenguajes formales como los naturales asocian un significado o **semántica** a cada cadena válida. Por ejemplo, en el lenguaje de la aritmética podemos tener una regla que diga que si «X» e «Y» son expresiones, entonces «$X + Y$» es también una expresión y su semántica es la suma de X e Y. En los lenguajes naturales es además importante entender la **pragmática** de una cadena: el significado real de la cadena cuando es dicha en una situación determinada. El significado no está solamente en las palabras en sí mismas, sino en la interpretación de las palabras *in situ*.

PRAGMÁTICA

ESTRUCTURA DE LA FRASE

La mayoría de los formalismos para reglas gramaticales están basados en la idea de **estructura de la frase** (que las cadenas están compuestas de subcadenas llamadas **frases**, las cuales pueden pertenecer a distintas categorías). Por ejemplo, las frases «el *wumpus*», «el rey» y «el agente de la esquina» son todas ellas ejemplos de la categoría **frase nominal** o *FN*. Hay dos razones para identificar a las frases de esta manera. Primera, las frases normalmente corresponden a elementos semánticos naturales a partir de los cuales puede ser interpretado el significado de una declaración; por ejemplo, las fra-

FRASE SUSTANTIVA

ses nominales hacen referencia a objetos del mundo. Segundo, el categorizar las frases nos ayuda a describir las cadenas permitidas del lenguaje. Podemos decir que cualquier frase nominal puede combinarse con una **frase verbal** (o *FV*), tal como «está muerto», para formar una frase de la categoría **sentencia** (o *S*). Sin las nociones intermedias de frase nominal y frase verbal sería difícil explicar por qué «el *wumpus* está muerto» es una sentencia mientras que «*wumpus* el muerto es» no lo es.

FRASE VERBAL

SENTENCIA

A los nombres de las categorías, tales como *FN*, *FV* y *S*, les llamamos **símbolos no terminales**. Las gramáticas definen los no terminales por medio de **reglas de reescritura**. Adoptaremos la notación de la forma Backus-Naur (BNF) para las reglas de reescritura, tal como se describe en el Apéndice B en la página 1.117. En esta notación, el significado de una regla como

SÍMBOLOS NO TERMINALES

REGLAS DE REESCRITURA

$$S \rightarrow FN \ FV$$

es que una sentencia puede consistir en una *FN* seguida por cualquier *FV*.

Etapas de la comunicación

Un episodio típico de comunicación, en el cual un hablante *H* quiere informar a un oyente *O* acerca de una sentencia *S* utilizando palabras *P*, está compuesto de siete procesos:

INTENCIÓN

Intención. Por alguna razón, un hablante *H* decide que hay una sentencia *S* que merece ser dicha al oyente *O*. En nuestro ejemplo, el hablante tiene la intención de hacer saber al oyente que el *wumpus* no vivirá más.

GENERACIÓN

Generación. El hablante planifica cómo convertir la sentencia *S* en una declaración que probablemente hará que el oyente, al percibir la declaración en la situación actual, pueda inferir el significado *S* (o algo parecido). Se asume que el hablante es capaz de proponer las palabras «El *wumpus* está muerto» y decirlas.

SÍNTESIS

Síntesis. El hablante produce la realización física P' de las palabras *P*. Esto puede ser vía tinta sobre papel, vibraciones en el aire o cualquier otro medio. En la Figura 22.1, mostramos un agente que sintetiza una cadena de sonidos P' escritos en el alfabeto fonético definido en el Apartado 15.6: «[elwumpusestamuerto]». Las palabras están juntas; esto es típico del habla rápida.

PERCEPCIÓN

Percepción. *O* percibe la realización física P' como P'_2 y la decodifica como las palabras P_2. Cuando el medio es hablado, el paso de percepción se llama **reconocimiento del habla**; cuando es impreso, se llama **reconocimiento óptico de caracteres**. Ambos pasaron en los años 1990 de ser esotéricos a ser temas comunes, debido en gran parte a la creciente potencia de cálculo de los computadores.

ANÁLISIS

Análisis. *O* infiere que P_2 tiene posibles significados $S_1, \ldots, S_n$. Dividimos el análisis en tres partes principales: interpretación sintáctica (o análisis sintáctico), interpretación semántica e interpretación pragmática. El **análisis sintáctico** es el proceso de construir el **árbol sintáctico** para una cadena de entrada, como se muestra en la Figura 22.1. Los nodos interiores del árbol sintáctico representan frases y los nodos hoja representan palabras. La **interpretación semántica** es el proceso de extracción del significado de una declaración como una expresión en algún lenguaje de representación. La Figura 22.1 muestra dos posibles interpretaciones semánticas: que el *wumpus* no está vivo y que el *wumpus* está agotado (un significado coloquial de *muerto*). Se dice que las declaraciones con varias interpretaciones posibles son **ambiguas**. La **interpretación pragmática** tiene en

ANÁLISIS SINTÁCTICO

ÁRBOL SINTÁCTICO

INTERPRETACIÓN SEMÁNTICA

INTERPRETACIÓN PRAGMÁTICA

CAPACIDAD GENERATIVA

Los formalismos gramaticales pueden ser clasificados por su **capacidad generativa**: el conjunto de lenguajes que pueden representar. Chomsky (1957) describe cuatro clases de formalismos gramaticales que difieren únicamente en la forma de las reglas de reescritura. Las clases pueden ser organizadas jerárquicamente, donde cada clase puede ser usada para describir todos los lenguajes que pueden ser descritos por una clase menos potente, así como algunos lenguajes adicionales. Vamos a listar la jerarquía; aparece primero la clase más potente:

Las **gramáticas recursivamente enumerables** usan reglas que no tienen restricciones: ambos lados de las reglas de reescritura pueden tener cualquier número de símbolos terminales y no terminales, como en la regla $A\ B \rightarrow C$. Estas gramáticas son equivalentes a las máquinas de Turing en cuanto a su potencia expresiva.

Las **gramáticas dependientes del contexto** están restringidas únicamente a que la parte derecha debe contener al menos tantos símbolos como la parte izquierda. El nombre «dependiente del contexto» procede del hecho de que una regla tal como $A\ S\ B \rightarrow A\ X\ B$ dice que un S puede ser reescrito como un X en el contexto de un A precediéndole y un B siguiéndole. Las gramáticas dependientes del contexto pueden representar lenguajes tales como $a^n\ b^n\ c^n$ (una secuencia de n copias de a seguida por el mismo número de b y de c).

En las **gramáticas independientes del contexto** (o **GIC**), el lado izquierdo consta de un único símbolo no terminal. Así, cada regla permite rescribir el no terminal como el lado derecho en *cualquier* contexto. Las GIC son muy utilizadas como gramáticas de lenguajes de programación y lenguajes naturales, aunque ahora está ampliamente aceptado que al menos algunos lenguajes naturales tienen construcciones que no son independientes del contexto (Pullum, 1991). Las gramáticas independientes del contexto pueden representar $a^n\ b^n$, pero no $a^n\ b^n\ c^n$.

Las **gramáticas regulares** son la clase más restringida. Cada regla tiene un único no terminal en el lado izquierdo y un símbolo terminal opcionalmente seguido por un no terminal en el lado derecho. Las gramáticas regulares son equivalentes en potencia a las máquinas de estados finitos. Se ajustan mal a los lenguajes de programación, debido a que no pueden representar construcciones tales como aperturas y cierres balanceados de paréntesis (una variación del lenguaje $a^n\ b^n$). Lo más cercano que pueden llegar es representando $a^*\ b^*$, una secuencia de cualquier número de a seguido de cualquier número de b.

Las gramáticas más altas en la jerarquía tienen más poder expresivo, pero los algoritmos para tratar con ellas son menos eficientes. Hasta mitad de los 80, los lingüistas se concentraron en los lenguajes independientes del contexto y dependientes del contexto. Desde entonces, ha crecido la importancia de las gramáticas regulares, ocasionado por la necesidad de procesar muy rápidamente megabytes y gigabytes de texto *en línea*, incluso a costa de un menor análisis completo. Como Fernando Pereira expresó, «Cuanto más viejo me vuelvo, más abajo voy en la jerarquía de Chomsky». Para ver lo que él quiere decir, compare Pereira y Warren (1980) con Mohri, Pereira y Riley (2002).

cuenta el hecho de que las mismas palabras pueden tener diferentes significados en diferentes situaciones. Mientras la interpretación sintáctica es una función de un argumento, la cadena, la interpretación pragmática es función de la declaración y del contexto o situación en la cual es enunciada. En el ejemplo, la pragmática hace dos cosas: reem-

Figura 22.1 Los siete procesos implicados en la comunicación, usando la sentencia ejemplo «El *wumpus* está muerto»

plaza la constante *Ahora* por la constante S_3, que representa la situación actual, y reemplaza *Wumpus* por $Wumpus_1$, que representa al único *wumpus* que sabemos que está en la cueva. En general, la pragmática puede contribuir mucho más a la interpretación final de la declaración; consideremos «Estoy mirando el diamante» cuando es dicho por un joyero o por un jugador de béisbol. En el Apartado 22.7 veremos que la pragmática nos permite interpretar «Está muerto» como que el *wumpus* está muerto si nosotros estamos en una situación donde el *wumpus* está saliente.

DESAMBIGUACIÓN

Desambiguación. *O* infiere que *H* intenta expresar S_i (donde idealmente $S_i = S$). La mayoría de los hablantes no son intencionalmente ambiguos, pero muchas declaraciones tienen varias interpretaciones factibles. La comunicación funciona porque el oyente hace la labor de imaginarse qué interpretación es la que probablemente pretende expresar el hablante. Conviene destacar que esta es la primera vez que hemos usado la palabra *probablemente*, y que la desambiguación es el primer proceso que depende fuertemente del razonamiento incierto. El análisis genera interpretaciones posibles; si se encuentra más de una interpretación, entonces la desambiguación elige aquella que considera la mejor.

INCORPORACIÓN

Incorporación. *O* decide creer S_i (o no). Un agente totalmente ingenuo deberá creer todas las cosas que oiga, pero un agente sofisticado considerará el acto de habla como evidencia para S_i, no como confirmación de ello.

Al ponerlo todo junto, obtenemos el programa agente mostrado en la Figura 22.2. Aquí el agente actúa como un robot esclavo que puede ser controlado por un amo. En cada turno, el esclavo responderá una pregunta o cumplirá una orden que le haya dado

```
función AGENTE-COMUNICATIVO-INGENUO (percepción) devuelve acción
    estático: BD, base de datos
              estado, el estado actual del retorno
              acción, la acción más reciente, inicialmente ninguna

estado ← ACTUALIZAR-ESTADO(estado, acción, percepción)
palabras ← PARTE-HABLADA(percepción)
semántica ← DESAMBIGUACIÓN(PRAGMÁTICA(SEMÁNTICA(ANALIZADOR(palabras))))
si palabras = Ninguna y no se dice acción entonces /* Describe el estado */
    devuelve DECIR(GENERAR-DESCRIPCIÓN(estado))
sino si TIPO[semántica] = Orden entonces /* Obedece la orden */
    devuelve CONTENIDOS[semántica]
sino si TIPO[semántica] = Pregunta entonces /* Contesta la pregunta */
    respuesta ← PREGUNTAR(BD, semántica)
    devuelve DECIR(GENERAR-DESCRIPCIÓN(respuesta))
sino si TIPO[semántica] = Sentencia entonces /* Cree la sentencia */
    NOTAR(BD, CONTENIDOS[semántica])
/* Si se fracasa aquí, hacer una acción «regular» */
    devuelve PRIMERO(PLANIFICADOR(BD, estado))
```

Figura 22.2 Un agente comunicativo que acepta órdenes, preguntas y sentencias. El agente puede también describir el estado actual o realizar una acción que no sea un acto de habla «regular» cuando no tiene nada que decir.

el amo y creerá cualquier afirmación hecha por el amo. También comentará la situación actual si no tiene nada más urgente que hacer y planificará su propia acción si se queda solo. Aquí tenemos un diálogo típico:

ROBOT ESCLAVO	AMO
Siento una brisa.	Ve a 1 2.
Aquí no hay nada.	Ve al norte.
Siento una brisa, huelo un hedor y veo un resplandor.	Toma el oro.

22.2 Una gramática formal para un fragmento del español

En esta sección definimos una gramática formal para un pequeño fragmento del inglés (castellano) que es adecuado para hacer afirmaciones sobre el mundo del *wumpus*. A este lenguaje le llamaremos ε_0. Secciones posteriores mejorarán ε_0 para hacerlo algo más cercano al español real. Es improbable idear una gramática completa para el español, aunque sólo sea porque dos personas no se pondrían totalmente de acuerdo sobre lo que constituye español válido.

El léxico de ε_0

LÉXICO

Primero definiremos el **léxico** o lista de palabras permitidas. Las palabras se agrupan en categorías o partes del habla conocidas por los usuarios del diccionario: sustantivos, pronombres y nombres para denotar cosas, verbos para denotar sucesos, adjetivos para modificar sustantivos y adverbios para modificar verbos. Las categorías que quizá sean menos familiares para algunos lectores son los artículos (tales como *el*), preposiciones (*en*) y conjunciones (*y*). La Figura 22.3 muestra un pequeño léxico.

Cada una de las categorías finaliza con … para indicar que hay otras palabras en la categoría. Sin embargo, debe hacerse notar que hay dos razones distintas para la falta de palabras. En principio no es factible listar todos los sustantivos, verbos, adjetivos y adverbios. No hay decenas de miles de miembros en cada clase, sino porque constantemente son añadidos otros nuevos (como *MP3* o *anime*). Estas cuatro categorías son llamadas **clases abiertas**. Las otras categorías (pronombres, artículos, preposiciones y conjunciones) son llamadas **clases cerradas**. Estas tienen un pequeño número de palabras (desde unas pocas a unas pocas docenas) que pueden en principio ser enumeradas completamente. Las clases cerradas cambian con el paso de los siglos, no de meses. Por ejemplo, «thee» y «thou»[2] fueron pronombres habitualmente usados en el siglo XVII, fueron decayendo en el siglo XIX y hoy en día sólo son vistos en poesía y algunos dialectos regionales.

CLASES ABIERTAS

CLASES CERRADAS

Sustantivo	→	**hedor** | **brisa** | **resplandor** | **nada** | **agente** | ***wumpus*** | **foso** | **oro** | **este** | …
Verbo	→	**es** | **ver** | **oler** | **disparar** | **sentir** | **heder** | **ir** | **tomar** | **llevar** | **matar** | **girar** | …
Adjetivo	→	**derecho** | **izquierdo** | **oriental** | **muerto** | **posterior** | **hediondo** | …
Adverbio	→	**aquí** | **allí** | **cerca** | **adelante** | **correctamente** | **a la izquierda** | **al este** | **al sur** | **atrás** | …
Pronombre	→	**mí** | **tú** | **yo** | **ello** | …
Nombre	→	**Juan** | **María** | **Alicante** | **Aristóteles** | …
Artículo	→	**el** | **un** | …
Preposición	→	**a** | **en** | **sobre** | …
Conjunción	→	**y** | **o** | **pero** | …
Dígito	→	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Figura 22.3 El léxico para ε_0.

La Gramática de ε_0

El siguiente paso es combinar las palabras en frases. Usaremos cinco símbolos no terminales para definir las diferentes clases de frases: sentencia (*S*), frase nominal (*FN*), frase verbal (*FV*), frase preposicional (*FP*) y oración relativa (*OracionRel*)[3]. La Figu-

[2] *Nota del traductor:* ti y tú en inglés antiguo.

[3] Una oración relativa sigue y modifica a una frase nominal. Consta de un pronombre relativo (tal como «quien» o «que») seguido por una frase verbal. (En el Ejercicio 22.12 se discute otra clase de oración relativa.) Un ejemplo de oración relativa es *que huele mal* en «El *wumpus* que huele mal está en 2 2».

S	→	*FN FV*	Yo + siento una brisa
	|	*S Conjunción S*	Yo siento una brisa + y + yo huelo a *wumpus*
FN	→	*Pronombre*	Yo
	|	*Nombre*	Juan
	|	*Sustantivo*	foso
	|	*Artículo* del *sustantivo*	el + *wumpus*
	|	*Dígito Dígito*	3 4
	|	*FN FP*	el *wumpus* + al este
	|	*FN OracionRel*	el *wumpus* + que es hediondo
FV	→	*Verbo*	huele mal
	|	*FV FN*	siento + una brisa
	|	*FV Adjetivo*	es hediondo
	|	*FV FP*	gira + al este
	|	*FV Advervio*	ve + adelante
FP	→	*Preposición FN*	a + el este
OracionRel	→	**que** *FV*	que + es hediondo

Figura 22.4 La gramática para $\mathcal{E}_0$, con frases de ejemplo para cada regla.

ra 22.4 muestra una gramática para ε_0, con un ejemplo para cada regla de reescritura. ε_0 genera sentencias correctas de español como:

Juan está en el foso
El *wumpus* que huele mal está en 2 2
María está en Boston y Juan huele mal

SOBREGENERA

SUBGENERACIÓN

Desafortunadamente, la gramática **sobregenera**: esto es, genera sentencias que no son gramaticales, tales como «Mi ir Boston» y «Yo huelo foso oro *wumpus* nada este». También **subgenera** hay muchas sentencias de inglés (castellano) que rechaza, tales como «Yo pienso que el *wumpus* es hediondo». (Otra deficiencia es que la gramática no pone en mayúsculas la primera palabra de una sentencia, ni añade puntuación al final. Esto es porque está diseñada principalmente para hablar, no para escribir.)

22.3 Análisis sintáctico

Hemos definido anteriormente el **análisis sintáctico** como el proceso de encontrar un árbol sintáctico para una cadena de entrada dada. Esto es, una llamada a la función ANÁLISISSINTÁCTICO, tal que

ANÁLISISSINTÁCTICO («el *wumpus* está muerto», ε_0, *S*)

debe devolver un árbol sintáctico con raíz *S* cuyas hojas son «el *wumpus* está muerto» y cuyos nodos internos son símbolos no terminales de la gramática ε_0. Se puede ver dicho árbol en la Figura 22.1. En texto lineal, escribimos el árbol como

[*S*: [*FN*: [*Artículo*: **el**][*Sustantivo*: ***wumpus***]]
[*FV*: [*Verbo*: **está**][*Adjetivo*: **muerto**]]]

El análisis sintáctico puede ser visto como un proceso de búsqueda del árbol sintáctico. Hay dos formas extremas de especificar el espacio de búsqueda (y muchas variantes entre ellas). En la primera, podemos empezar con símbolo *S* y buscar un árbol que tenga las palabras en sus hojas. Este modo se llama **análisis descendente** (ya que *S* es dibujado en lo alto del árbol). En la segunda, podemos empezar con las palabras y buscar un árbol con raíz *S*. Este modo se llama **análisis ascendente**[4]. El análisis descendente puede ser definido precisamente como un problema de búsqueda tal que:

ANÁLISIS ASCENDENTE

ANÁLISIS DESCENDENTE

- El **estado inicial** es un árbol sintáctico que consta de la raíz *S* y un hijo desconocido: [*S*: ?]. En general, cada estado del espacio de búsqueda es un árbol sintáctico.
- La **función sucesor** selecciona el nodo más a la izquierda del árbol con hijo desconocido. Entonces busca en la gramática reglas que tengan la etiqueta raíz del nodo en el lado izquierdo. Por cada una de estas reglas, crea un estado sucesor donde el ? es reemplazado por la lista correspondiente al lado derecho de la regla. Por ejemplo, en ε_0 hay dos reglas para *S*, así que el árbol [*S*: ?] podría ser reemplazado por los dos sucesores siguientes:

  [*S*: [*S*: ?][*Conjunción*: ?][*S*: ?]]
  [*S*: [*FN*: ?][*FV*: ?]]

  La segunda de estas tiene siete sucesores, uno por cada regla de reescritura de *FN*.
- La **prueba (criterio) de finalización** controla que las hojas del árbol sintáctico correspondan exactamente a la cadena de entrada, sin entradas desconocidas ni descubiertas.

Un gran problema para el análisis descendente es hacer frente a las llamadas **reglas recursivas por la izquierda**, esto es, reglas de la forma $X \rightarrow X$... Con la búsqueda primero en profundidad, dichas reglas podrían llevarnos a seguir reemplazando *X* con [*X*: *X*…] en un bucle infinito. Con la búsqueda de primero en anchura podríamos encontrar exitosamente árboles sintácticos para sentencias válidas, pero cuando se nos da una sentencia inválida, podríamos quedar atascados en un espacio de búsqueda infinito.

La formulación del análisis ascendente como una búsqueda es como sigue:

- El **estado inicial** es una lista de las palabras de la cadena de entrada, cada una vista como un árbol sintáctico consistente en un único nodo hoja, por ejemplo [**el**, ***wumpus***, **está**, **muerto**]. En general, cada estado del espacio de búsqueda es una lista de árboles sintácticos.

[4] El lector debe darse cuenta de que el análisis descendente y el ascendente son análogos al encadenamiento hacia atrás y hacia delante, respectivamente, descritos en el Capítulo 7. Veremos brevemente que la analogía es exacta.

- La **función sucesor** mira en cada posición *i* de la lista de árboles y en cada lado derecho de las reglas de la gramática. Si la subsecuencia de la lista de árboles que empieza en *i* empareja con el lado derecho, entonces la subsecuencia es reemplazada por un nuevo árbol cuya categoría es el lado izquierdo de la regla y cuyo hijo es la subsecuencia. Por «emparejar» queremos decir que la categoría del nodo es la misma que el elemento del lado derecho. Por ejemplo, la regla *Artículo* → **el** empareja la subscuencia que consta del primer nodo de la lista [**el**, ***wumpus***, **está**, **muerto**], así que un estado sucesor podría ser [[*Artículo*: **el**], ***wumpus***, **está**, **muerto**].
- La **prueba (criterio) de finalización** comprueba el estado que conste de un árbol sencillo con raíz *S*.

En la Figura 22.5 podemos ver un ejemplo de un análisis ascendente.

paso	*lista de nodos*	*subsecuencia*	*regla*
INIT	**el *wumpus* está muerto**	**el**	*Artículo* → **el**
2	*Artículo* ***wumpus*** **está muerto**	***wumpus***	*Sustantivo* → ***wumpus***
3	*Artículo Sustantivo* **está muerto**	*Artículo Sustantivo*	*FN* → *Artículo Sustantivo*
4	*FN* **está muerto**	**está**	*Verbo* → **está**
5	*FN Verbo* **muerto**	**muerto**	*Adjetivo* → **muerto**
6	*FN Verbo Adjetivo*	*Verbo*	*FV* → *Verbo*
7	*FN FV Adjetivo*	*FV Adjetivo*	*FV* → *FV Adjetivo*
8	*FN FV*	*FN FV*	*S* → *FN FV*
OBJETIVO	*S*		

Figura 22.5 Traza de un análisis ascendente de la cadena «El *wumpus* está muerto». Empezamos con una lista de nodos consistentes en palabras. Después reemplazamos subsecuencias que emparejan el lado derecho de una regla con un nuevo nodo cuya raíz es el lado izquierdo. Por ejemplo, en la tercera línea los nodos *Artículo* y *Sustantivo* son reemplazados por un nodo *FN* que tiene estos dos nodos como hijos. El análisis descendente produciría una traza similar, pero en la dirección opuesta.

Tanto el análisis ascendente como el descendente pueden ser ineficientes, debido a la multiplicidad de maneras en las cuales múltiples árboles sintácticos para diferentes frases pueden ser combinados. Ambos pueden malgastar tiempo buscando irrelevantes porciones del espacio de búsqueda. El análisis descendente puede generar nodos intermedios que nunca podrán ser sujetados por palabras, y el análisis ascendente puede generar análisis parciales de las palabras que no pueden aparecer en un *S*.

Incluso si tuviésemos una función heurística perfecta que nos permitiera buscar sin ninguna digresión irrelevante, estos algoritmos podrían todavía ser ineficientes, ya que algunas sentencias tienen *exponencialmente muchos* árboles sintácticos. La siguiente subsección muestra qué hacer en estos casos.

Análisis sintáctico eficiente

Consideremos las siguientes dos sentencias[5]:

> *Have the students in section 2 of Computer Science 101 take the exam.*
> (Los estudiantes de la sección 2 de Ciencia de la Computación 101 han realizado el examen.)
> *Have the students in section 2 of Computer Science 101 taken the exam?*
> (¿Han realizado el examen los estudiantes de la sección 2 de Ciencia de la Computación 101?)

Incluso aunque comparten las 10 primeras palabras, estas sentencias tiene análisis sintácticos muy diferentes, ya que la primera es una orden y la segunda una pregunta. Un algoritmo de análisis sintáctico de izquierda a derecha tendría que suponer si la primera palabra es parte de una orden o de una pregunta y no podría decir si la suposición es correcta al menos hasta la undécima palabra, *take* o *taken*. Si el algoritmo supone equivocadamente, tendrá que deshacer todo el camino hasta la primera palabra. Esta clase de vuelta atrás es inevitable, pero si nuestro algoritmo de análisis sintáctico pretende ser eficiente, debe evitar reanalizar «the students in section 2 of Computer Science 101» como un *FN* cada vez que retroceda.

En esta sección, desarrollaremos un algoritmo de análisis sintáctico que evita esta fuente de ineficiencias. La idea básica es un ejemplo de **programación dinámica**: *cada vez que analizamos una subcadena, almacenamos el resultado de forma que no tenemos que reanalizarla posteriormente*. Por ejemplo, una vez que descubrimos que «the students in section 2 of Computer Science 101» es una *FN*, podemos registrar este resultado en una estructura de datos conocida como un **grafo**. Los algoritmos que hacen esto son llamados **analizadores sintácticos con grafos**. Debido a que estamos tratando con gramáticas independientes del contexto, cualquier frase que se encontrase en el contexto de una rama del espacio de búsqueda puede funcionar igualmente bien en cualquier otra rama del espacio de búsqueda.

GRAFO

El grafo para una sentencia de n palabras consta de $n + 1$ **nodos** y un número de **arcos** que conectan nodos. La Figura 22.6 muestra un grafo con seis nodos (círculos) y tres arcos (líneas). Por ejemplo, el arco etiquetado

NODOS

ARCOS

$$[0,5, S \rightarrow FN\ FV\ \bullet]$$

significa que una *FN* seguida por una *FV* se combinan para formar una *S* que cruza la cadena desde 0 hasta 5. El símbolo • en un arco separa lo que ha sido encontrado hasta el momento de lo que queda por encontrar[6]. Los arcos con • al final son llamados **arcos completos**. El arco

$$[0,2, S \rightarrow FN \bullet FV]$$

[5] *Nota del traductor:* mantendremos, en este caso y otros posteriores, las frases del original inglés para poder seguir el argumento del autor, en este caso de la coincidencia de las palabras, ya que en muchos casos hacen referencia a particularidades de la gramática inglesa.

[6] Debido al • los arcos son llamados a veces **reglas punteadas.**

Figura 22.6 Parte del grafo para la sentencia «El agente siente una brisa». Se muestran los seis nodos, pero solamente tres de los arcos que pueden componer un análisis sintáctico completo.

dice que una *FN* cruza la cadena de 0 a 2 (las dos primeras palabras) y que si podemos encontrar a continuación una *FV*, entonces podemos formar una *S*. Los arcos como este con el punto antes del final son llamados arcos incompletos, y decimos que el arco está buscando una *FV*.

La Figura 22.7 muestra un algoritmo de análisis sintáctico con grafo. La idea principal es combinar lo mejor del análisis sintáctico ascendente y del descendente. El procedimiento PREDECIR descendente: transforma las entradas en el grafo que dicen qué símbolos son deseables y en qué localizaciones. ANALIZAR es el procedimiento ascendente que se inicia en las palabras, pero que utilizará una única palabra para extender una entrada de un grafo ya existente. De forma similar, EXTENDER construye componentes descendentes, pero solamente para extender una entrada de un grafo ya existente.

Usamos un truco para iniciar el algoritmo completo: añadimos el arco $[0, 0, S' \rightarrow \bullet\, S]$ al grafo, donde S es el símbolo inicial de la gramática, y S' es un nuevo símbolo que acabamos de idear. La llamada a AÑADIR-ARCO causa que el PREDECIR añada arcos por las reglas que puedan producir un S, esto es, $[S \rightarrow FN\, FV]$. Después miramos el primer componente de esa regla, *FN*, y añadimos reglas para cada manera de producir una *FN*. Finalmente, el predictor (pronosticador) añade, en forma descendente, todos los posibles arcos que pueden ser usados para crear la S final.

Cuando el predictor para S' finaliza, entramos en un bucle que llama a SCANNER para cada palabra de la sentencia. Si la palabra en la posición j es miembro de la categoría B que algún arco está buscando en j, entonces extendemos ese arco, anotando la palabra como una instancia de B. Observemos que cada llamada a ANALIZAR puede acabar llamando a PREDECIR y EXTENDER recursivamente, de ese modo intercalar procesamiento descendente y ascendente.

El otro componente ascendente, EXTENDER[7], toma un arco completo con lado izquierdo B y lo usa para extender cualquier regla incompleta del grafo que finaliza donde el arco completo empieza si la regla incompleta está buscando a B.

Las Figuras 22.8 y 22.9 muestran un grafo y una traza del algoritmo analizando la sentencia «Yo la percibo» («I feel it») (que es una respuesta a la pregunta «¿Sientes una

[7] Tradicionalmente nuestro procedimiento EXTENDER ha sido llamado COMPLETER. Este nombre es engañoso, ya que el procedimiento no completa arcos: toma un arco completo como entrada y extiende los arcos incompletos.

```
función ANALIZADOR-CON-GRAFO(palabras, gramática) devuelve grafo
    grafo ← array[0... LONGITUD(palabras)] de listas vacias
    AÑADIR-ARCO  S([0, 0, S′ → • S])
    para i desde 0 a LONGITUD(palabras) hacer
        ANALIZAR(i, palabras[i])
    devuelve grafo

procedimiento AÑADIR-ARCO(arco))
    /* Añadir un arco al grafo y ver si extiende o predice otro arco. */
    si arco no está en grafo[FIN(arco)] entonces
        añadir arco a el grafo[FIN(arco)]
        si arco no tiene nada después del punto entonces EXTENDER(arco)
        sino PREDECIR(arco)

procedimiento ANALIZAR(j, word)
    /* Para cada arco que espera una palabra de esta categoría, se extiende el arco. */
    para cada [i, j, A → α • B β] en grafo[ j] hacer
        si palabra es de la categoría B entonces
            AÑADIR-ARCO([i, j + 1, A → α B • β])

procedimiento PREDECIR([i, j, A → α • B β])
    /* Añadir al grafo todas las reglas de B que podrían ayudar a extender el arco */
    para cada (B → γ) en REESCRIBIR-PARA(B, gramática) hacer
        AÑADIR-ARCO([j, j, B → • γ])

procedimiento EXTENDER([j, k, B → γ •])
    /* Ver qué arcos se pueden extender mediante este arco */
    e_B ← el arco que es la entrada a este procedimiento
    para cada ([i, j, A → α • B′ β]) en grafo[j] hacer
        si B = B′ entonces
            AÑADIR-ARCO([i, j, A → α e_B • β])
```

Figura 22.7 El algoritmo de análisis sintáctico con grafo. S es el símbolo inicial y S' es un símbolo no terminal nuevo. ***grafo[j]*** es la lista de los arcos que finalizan en el nodo j. Las letras griegas equivalen a una cadena de cero o más símbolos.

brisa?»). Trece arcos (etiquetados a-m) son registrados en el grafo, incluyendo cinco arcos completos (mostrados sobre los nodos del grafo) y ocho incompletos (debajo de los nodos). Observar el ciclo de las acciones predecir, analizar y extender. Por ejemplo, predictor utiliza el hecho de que el arco (a) está buscando una *S* para autorizar la predicción de una *FN* (arco b) y después un *Pronombre* (arco c). Entonces analizar reconoce que hay un *Pronombre* en la posición derecha (arco d) y extender combina el arco incompleto b con el arco completo d para producir un nuevo arco, e.

El algoritmo de análisis sintáctico con grafo evita construir una extensa clase de arcos que podrían haber sido examinados por el procedimiento ascendente simple. Consideremos la sentencia «The ride the horse gave was wild» («La carrera que el caballo realizó fue desenfrenada»). Un análisis sintáctico ascendente podría etiquetar «ride the horse» como una *FV* y después descartar el árbol sintáctico cuando encontrara que no po-

Figura 22.8 Grafo para el análisis sintáctico de «$_0$ Yo $_1$ lo $_2$ siento $_3$» («$_0$ I $_1$ feel $_2$ it $_3$»). La notación *m:S* significa que el arco *m* tiene una *S* en el lado izquierdo, mientras que la notación *f:FV/Verbo* significa que el arco *f* tienen una *FV* en el lado izquierdo, pero está buscando un *Verbo*. Hay cinco arcos completos sobre los nodos y ocho arcos incompletos debajo de ellos.

Arco	Procedimiento	Derivación
a	INICIAR	[0, 0, $S' \rightarrow \bullet\ S$]
b	PREDECIR(a)	[0, 0, $S \rightarrow \bullet\ FN\ FV$]
c	PREDECIR(b)	[0, 0, $FN \rightarrow \bullet\ Pronombre$]
d	ANALIZAR(c)	[0, 1, $FN \rightarrow Pronombre\ \bullet$]
e	EXTENDER(b,d)	[0, 1, $S \rightarrow FN \bullet FV$]
f	PREDECIR(e)	[1, 1, $FV \rightarrow \bullet\ Verbo$]
g	PREDECIR(e)	[1, 1, $FV \rightarrow \bullet\ FV\ FN$]
h	ANALIZAR(f)	[1, 2, $FV \rightarrow Verbo\ \bullet$]
i	EXTENDER(g,h)	[1, 2, $FV \rightarrow FV \bullet FN$]
j	PREDECIR(g)	[2, 2, $FN \rightarrow \bullet\ Pronombre$]
k	ANALIZAR(j)	[2, 3, $FN \rightarrow Pronombre\ \bullet$]
l	EXTENDER(i,k)	[1, 3, $FV \rightarrow FV\ FN\ \bullet$]
m	EXTENDER(e,l)	[0, 3, $S \rightarrow FN\ FV\ \bullet$]

Figura 22.9 Traza del análisis sintáctico de «$_0$ Yo $_1$ lo $_2$ siento $_3$» («$_0$ I $_1$ feel $_2$ it $_3$»). Para cada arco a-m, mostramos el procedimiento usado para derivar el arco desde otros arcos ya existentes en el grafo. Por brevedad omitimos algunos arcos.

día encajarlo en una *S* más amplia. Pero ε_0 no permite que una *FV* siga a «el», así que el algoritmo de análisis sintáctico con grafo nunca predecirá una *FV* en este punto y por tanto evitará perder tiempo construyendo la componente *FV* aquí. Los algoritmos que tra-

EXTREMO IZQUIERDO

bajan de izquierda a derecha y evitan construir estos componentes imposibles son llamados analizadores sintácticos de **extremo izquierdo**, debido a que montan el árbol sintáctico que empieza con el elemento inicial de la gramática y extienden la palabra más a la izquierda de la sentencia (el extremo izquierdo). Únicamente se añadirá un arco al grafo si puede servir para extender este árbol sintáctico. (*Véase* la Figura 22.10 como ejemplo).

El analizador sintáctico con grafo utiliza únicamente tiempo y espacio polinomial. Requiere espacio para almacenar arcos del orden $O(kn^2)$, donde n es el número de palabras de la sentencia y k es una constante que depende de la gramática. Cuando no puede construir más arcos, para, así sabemos que el algoritmo termina (incluso cuando hay reglas recursivas por la izquierda). De hecho, tarda de orden de $O(n^3)$ en el peor de los casos, lo cual es lo mejor que se puede conseguir para gramáticas independientes del contexto. El cuello de botella para ANALIZADOR-CON-GRAFO es EXTENDER, el cual debe intentar extender cada uno de los $O(n)$ arcos incompletos que finalizan en la posición j con cada uno de los $O(n)$ arcos completos que empiezan en j, para cada uno de los $n + 1$ valores diferentes de j. Multiplicándolos obtenemos $O(n^3)$. Esto nos da una especie de paradoja: ¿cómo puede un algoritmo de $O(n^3)$ devolver una respuesta que puede contener un número exponencial de árboles sintácticos? Consideremos un ejemplo: la sentencia

«Fall leaves fall and spring leaves spring»
(Las hojas otoñales caen y las hojas primaverales brotan)

es ambigua ya que cada palabra (excepto «and») puede ser tanto sustantivo como verbo, y «fall» y «spring» pueden ser también adjetivos. En total, la sentencia tiene cuatro análisis sintácticos[8]:

Figura 22.10 Un algoritmo de análisis sintáctico de extremo izquierdo evita predecir una *FV* empezando en «ride», pero no predice una *FV* empezando en «was», ya que la gramática espera que una *FV* siga a una *FN*. El triángulo sobre «the horse gave» significa que las palabras han pasado un análisis sintáctico con una *OraciónRel*, pero con componentes intermedios adicionales que no son mostrados.

[8] El análisis sintáctico [*S*: Fall [*FV*: leaves fall]] es equivalente a «Autumn abandons autumn» (El otoño abandona al otoño).

[*S*: [*S*: [*FN*: Fall leaves] fall] and [*S*: [*FN*: spring leaves] spring];
[*S*: [*S*: [*FN*: Fall leaves] fall] and [*S*: spring [*FV*: leaves spring]];
[*S*: [*S*: Fall [*FV*: leaves fall]] and [*S*: [*FN*: spring leaves] spring];
[*S*: [*S*: Fall [*FV*: leaves fall]] and [*S*: spring [*FV*: leaves spring]].

Si tuviésemos conjuntamente n subsentencias ambiguas, podríamos tener 2^n maneras de elegir análisis sintácticos para las subsentencias[9]. ¿Cómo debe el analizador sintáctico con grafo evitar un tiempo de procesamiento exponencial? Realmente hay dos respuestas. La primera es que el algoritmo CHART-PARSE en sí mismo es realmente un reconocedor, no un analizador. Si hay un arco completo de la forma $[0, n, S \rightarrow \alpha \bullet]$ en el grafo, entonces hemos reconocido una S. La recuperación del árbol sintáctico desde este arco no es considerado parte del trabajo de CHART-PARSE, pero puede hacerlo. Observar, en la última línea de EXTENDER, que montamos α como una lista de arcos, e_B, no como una lista de nombres de categorías. Así, para convertir un arco en un árbol sintáctico, simplemente miramos recursivamente a los arcos componentes, convirtiendo cada $[i, j, X \rightarrow \alpha \bullet]$ en el árbol $[X : \alpha]$. Esto es sencillo, pero nos da sólo *un* árbol sintáctico.

La segunda respuesta es que si quiere todos los posibles análisis sintácticos, tendrá que escarbar profundamente en el grafo. Mientras estamos convirtiendo el arco $[i, j, X \rightarrow \alpha \bullet]$ en el árbol $[X : \alpha]$, también miraremos para ver si hay otros arcos de la forma $[i, j, X \rightarrow \beta \bullet]$. Si los hay, estos arcos generarán análisis sintácticos adicionales. Ahora tenemos una selección de lo que podemos hacer con ellos. Podemos enumerar todas las posibilidades, y esto significa que la paradoja puede ser resuelta y podríamos necesitar una cantidad exponencial de tiempo para listar los análisis sintácticos. O podemos prolongar el misterio un poco más y representar los análisis sintácticos con una estructura llamada **bosque empaquetado**, la cual se parece a esto:

BOSQUE EMPAQUETADO

$$[S\text{: } [S\text{: } \left\{\begin{matrix}[FN\text{: Fall leaves}][FV\text{: fall}] \\ [FN\text{: Fall}][FV\text{: leaves fall}]\end{matrix}\right\}] \text{ and } [S\text{: } \left\{\begin{matrix}[FN\text{: spring leaves}][FV\text{: spring}] \\ [FN\text{: spring}][FV\text{: leaves spring}]\end{matrix}\right\}]]$$

La idea es que cada nodo pueda ser o bien un nodo de un árbol sintáctico regular o un conjunto de nodos árbol. Esto nos permitirá devolver una representación de un número exponencial de análisis sintácticos en una cantidad de tiempo y espacio polinomial. Desde luego, cuando $n = 2$, no hay muchas diferencias entre 2^n y $2n$, pero para n grande, esta representación proporciona un ahorro considerable. Desafortunadamente, esta simple aproximación de bosque empaquetado no manejará todas las $O(n!)$ ambigüedades de cómo se asocian las conjunciones. Maxwell y Kaplan (1995) muestran cómo una representación más compleja, basada en los principios de los sistemas de mantenimiento de la coherencia puede empaquetar los árboles aún más ajustadamente.

[9] También podría ser una ambigüedad $O(n!)$ en la manera en que los componentes se juntan entre ellos, por ejemplo, (X y (Y y Z)) frente ((X y Y) y Z). Pero esto es otra historia, una que está bastante bien explicada por Church y Patil (1982).

22.4 Gramáticas aumentadas

En el Apartado 22.2 vimos que una gramática simple para ε_0 genera «Yo huelo un hedor» («I smell a stench») y otras frases de inglés. Desafortunadamente, también genera muchas frases incorrectas como «Mi huelo un hedor» («Me smell a stench»). Para evitar este problema, nuestra gramática debería conocer que «me» no es un *FN* válida cuando es el sujeto de una frase. Los lingüistas dicen que el pronombre «Yo» se emplea en el caso subjetivo, y «mi» en el caso objetivo[10]. Cuando tenemos los casos en cuenta comprendemos que ε_0 no es una gramática independiente del contexto: no es verdad que cualquier *FN* sea igual a cualquier otra sin hacer caso del contexto. Podemos arreglar el problema introduciendo nuevas categorías tales como FN_S y FN_O, para soportar frases sustantivas en los casos subjetivo y objetivo, respectivamente. También necesitamos separar la categoría *Pronombre* en las dos categorías $Pronombre_S$ (que incluye «yo») y $Pronombre_O$ (que incluye «mi»). La parte superior de la Figura 22.11 muestra la gramática BNF completa para el caso convenido; podemos llamar al lenguaje resultante ε_1. Hay que reseñar que todas las reglas de las *FN* deben ser duplicadas, una vez para las FN_S y otra para las FN_O.

CONCORDANCIA

Desafortunadamente, ε_1 todavía sobregenera. El español y otros muchos lenguajes requieren **concordancia** entre el sujeto y el verbo principal de una frase. Por ejemplo, si «yo» es el sujeto, entonces «yo huelo» es gramaticalmente correcto, pero «yo huele» no lo es. Si «ello» es el sujeto, tenemos el caso inverso. En inglés, las distinciones por convenio son mínimas: la mayoría de los verbos tienen una forma para los sujetos singulares en tercera persona (él, ella o ello), y una segunda forma para todas las otras com-

$$
\begin{aligned}
S &\rightarrow FN_S\ FV \mid \ldots \\
FN_S &\rightarrow Pronombre_S \mid Nombre \mid Sustantivo \mid \ldots \\
FN_O &\rightarrow Pronombre_O \mid Nombre \mid Sustantivo \mid \ldots \\
FV &\rightarrow FV\ FN_O \mid \ldots \\
FP &\rightarrow Preposición\ FN_O \\
Pronombre_S &\rightarrow \mathbf{Yo} \mid \mathbf{tu} \mid \mathbf{el} \mid \mathbf{ella} \mid \mathbf{ello} \mid \ldots \\
Pronombre_O &\rightarrow \mathbf{mi} \mid \mathbf{tu} \mid \mathbf{su} \mid \ldots
\end{aligned}
$$

$$
\begin{aligned}
S &\rightarrow FN(Subjetivo)\ FV \mid \ldots \\
FN(caso) &\rightarrow Pronombre(caso) \mid Nombre \mid Sustantivo \mid \ldots \\
FV &\rightarrow FV\ FN(Objetivo) \mid \ldots \\
FP &\rightarrow Preposición\ FN(Objetivo) \\
Pronombre(Subjetivo) &\rightarrow \mathbf{Yo} \mid \mathbf{tu} \mid \mathbf{el} \mid \mathbf{ella} \mid \mathbf{ello} \mid \ldots \\
Pronombre(Objetivo) &\rightarrow \mathbf{mi} \mid \mathbf{tu} \mid \mathbf{su} \mid \ldots
\end{aligned}
$$

Figura 22.11 Arriba: una gramática en BNF para el lenguaje ε_1, la cual maneja casos objetivos y subjetivos en frases sustantivas y de esta forma no sobregenera demasiado incorrectamente. Se han omitido las porciones que son idénticas a ε_0. Abajo: una gramática de cláusulas definidas (DCG) de ε_1.

[10] El caso subjetivo también es a veces llamado el caso nominativo y el caso objetivo es llamado el caso acusativo. Muchos lenguajes también tienen un caso dativo para palabras en la posición del objeto indirecto.

binaciones de persona y número. Hay una excepción: «I am / you are / he is» tiene tres formas. Si multiplicamos estas tres distinciones por las dos distinciones FN_S y FN_O, acabamos con seis formas de *FN*. Como descubriremos más distinciones, acabaremos empleando números exponenciales.

AUMENTAR

La alternativa es **aumentar** las reglas de la gramática en vez de introducir nuevas reglas. Daremos primero un ejemplo de cómo queremos representar una regla aumentada. Las reglas aumentadas permiten *parámetros* para las categorías no terminales. La Figura 22.11 muestra cómo describir ε_1 empleando reglas aumentadas. Las categorías *FN* y *Pronombre* tienen un parámetro indicando su caso. (Los sustantivos no tienen caso en inglés, aunque lo tienen en muchos otros lenguajes.) En la regla para *S*, la *FN* debe estar en caso subjetivo, mientras que en las reglas para *FV* y *FP*, la *FN* debe estar en caso objetivo. La regla para *FN* toma una variable, *caso*, como su argumento. La intención es que la *FN* pueda tener cualquier caso, pero si la *FN* es rescrita como *Pronombre*, entonces ésta tenga el mismo caso. Este uso de una variable (evitando una decisión donde la distinción no es importante) es lo que mantiene el tamaño del conjunto de reglas evitando un crecimiento exponencial con el número de características.

GRAMÁTICA DE CLÁUSULAS DEFINIDAS

Este formalismo para aumentar es llamado **gramática de cláusulas definidas** o DCG (*definite clause grammar*), debido a que cada regla gramatical puede ser interpretada como una cláusula definida en la lógica de Horn[11]. Primero mostraremos cómo una regla normal, no aumentada, puede ser interpretada como una cláusula definida. Consideremos cada símbolo de categoría como un predicado sobre cadenas, de forma que *FN*(s) es verdad si la cadena *s* forma una *FN*. La regla CFG

$$S \rightarrow FN\ FV$$

es la forma reducida de la cláusula definida

$$FN(s_1) \wedge FV(s_2) \Rightarrow S(s_1 + s_2)$$

Aquí $s_1 + s_2$ denota la concatenación de dos cadenas, por tanto esta regla dice que si la cadena s_1 es una *FN* y la cadena s_2 es una *FV*, entonces su concatenación es una *S*, que es exactamente como estamos interpretando la regla CFG. Es importante hacer notar que dichas *DCG's nos permiten hablar del análisis sintáctico como inferencia lógica.* Esto hace posible razonar sobre lenguajes y cadenas de formas muy diversas. Por ejemplo, significa que podemos hacer un análisis sintáctico ascendente empleando encadenamiento hacia adelante o análisis sintáctico descendente empleando encadenamiento hacia atrás. Veremos que también esto significa que podemos usar la misma gramática tanto para el análisis sintáctico como para la generación.

El beneficio real del enfoque DCG es que podemos *aumentar* la categoría de símbolos con argumentos adicionales aparte de los argumentos de cadena. Por ejemplo, la regla

$$FN(caso) \rightarrow Pronombre(caso)$$

es la abreviatura de

$$Pronombre(caso, s_1) \Rightarrow FN(caso, s_1)$$

[11] Recordar que una cláusula definida, cuando se escribe como una implicación, tiene exactamente un átomo en su consecuente, y una conjunción de cero o más átomos en su antecedente. Dos ejemplos son $A \wedge B \Rightarrow C$ y sólo C.

Esta dice que si la cadena s_1 es un *Pronombre* con el caso especificado por la variable *caso*, entonces s_1 es también una *FN* con el mismo caso. En general, podemos aumentar la categoría de símbolos con cualquier número de argumentos, y los argumentos son parámetros que son susceptibles de unificarse como en la inferencia regular de cláusulas de Horn.

Hay que pagar un precio por esta ventaja: estamos proporcionando al escritor de gramáticas el poder completo de un demostrador de teoremas, por tanto debemos proporcionar las garantías de un análisis sintáctico de orden $O(n^3)$; el análisis sintáctico con aumentos puede ser NP-completo o incluso indecidible, dependiendo de los aumentos.

Unas cuantas argucias más son necesarias para hacer que la DCG funcione; por ejemplo, necesitamos una manera de especificar los símbolos terminales, y es conveniente tener una manera de *no* añadir automáticamente el argumento de cadena. Poniendo todo junto, definimos una gramática de cláusulas definidas como sigue:

- La notación $X \rightarrow Y\ Z \ldots$ se traduce como $Y(s_1) \wedge Z(s_2) \wedge \ldots \Rightarrow X(s_1 + s_2 + \ldots)$.
- La notación $X \rightarrow Y \mid Z \mid \ldots$ se traduce como $Y(s) \vee Z(s) \vee \ldots \Rightarrow X(s)$.
- En cualquiera de las reglas anteriores, cualquier símbolo Y puede ser aumentado con uno o más argumentos. Cada argumento puede ser una variable, una constante o una función de argumentos. En la traducción, estos argumentos preceden a la cadena de argumentos (por ejemplo, *FN*(*caso*) se traduce como *FN*(*caso*, s_1)).
- La notación $\{P(\ldots)\}$ puede aparecer en el lado derecho de una regla y se traduce palabra por palabra en $P(\ldots)$. Esto permite al escritor de gramáticas insertar una prueba para $P(\ldots)$ sin tener añadido el argumento de cadena automático.
- La notación $X \rightarrow$ **palabra** se traduce como $X([palabra])$.

El problema de la concordancia sujeto-verbo podría ser también manejado con aumentos, pero dejaremos esto para el Ejercicio 22.2. En vez de esto abordaremos un problema más difícil: la subcategorización del verbo.

Subcategorización del verbo

ε_1 es una mejora sobre ε_0, pero la gramática ε_1 todavía sobregenera. Un problema está en la manera en la que las frases verbales se colocan juntas. Queremos aceptar frases verbales como «dame el oro» y «ve a l 2». Todas ellas están en ε_1, pero desafortunadamente también lo están «ve me el oro» y «da a l 2». El lenguaje ε_2 elimina estas *FV* declarando explícitamente qué frases pueden seguir a qué verbos. Llamamos a esta lista la lista de **subcategorización** para el verbo. La idea es que la categoría *Verbo* se parte en subcategorías: una para verbos que no tienen objeto, otra para verbos que tiene un objeto único, y así.

SUBCATEGORIZACIÓN

LISTA DE SUBCATEGORIZACIÓN

COMPLEMENTOS

Para implementar esta idea, daremos a cada verbo una **lista de subcategorización** que enumere los **complementos** de los verbos. Un complemento es una frase obligatoria que sigue al verbo dentro de la frase verbal. Así en «Da el oro a mi», la *FN* «el oro» y el *FP* «a mi» son complementos de «da»[12]. Deberíamos escribir esto como

$$Verbo([FN, FP]) \rightarrow \textbf{dar} \mid \textbf{mano} \mid \ldots$$

[12] Esta es una definición de *complemento*, pero otros autores emplean una terminología diferente. Algunos dicen que el sujeto de un verbo es también un complemento. Otros dicen que una frase preposicional es un complemento y que la frase nominal debería llamarse **argumento**.

Es posible para un verbo tener varias subcategorizaciones diferentes, tal y como es posible para una palabra pertenecer a varias categorías diferentes. De hecho, «da» también tiene la lista de subcategorización [*FN, FP*], como en «Dame el oro». Podemos tratar esto como otro tipo de ambigüedad. La Figura 22.12 proporciona varios ejemplos de verbos y sus listas de subcategorización (o **subcat**. para abreviar).

Para integrar la subcategorización del verbo en la gramática, realizamos tres pasos. El primer paso es aumentar la categoría *FV* para que tome un argumento de subcategorización, *FV*(*subcat*.) que indica la lista de complementos que es necesaria para formar una *FV* completa. Por ejemplo, «da» puede ser convertido en una *FV* completa añadiendo [*FN, FP*], «da el oro» se puede hacer completa añadiendo [*FP*], y «da el oro a mí» es ya una *FV* completa. Por tanto su lista de subcategorización es la lista vacía, [ ]. Esto nos proporciona estas reglas para *FV*:

$$
\begin{aligned}
FV(subcat) \rightarrow\ & Verbo(subcat) \\
|\ & FV(subcat + [FN])\ FN(Objetivo) \\
|\ & FV\ (subcat + [Adjetivo])\ Adjetivo \\
|\ & FV\ (subcat + [FP])\ FP
\end{aligned}
$$

La última línea pueden ser leída como «Una *FV* con una lista de subcat, *subcat*, puede ser formada por una *FV* embebida seguida por una *FP*, así como la *FV* embebida tiene una lista de subcat que comienza con los elementos de la lista *subcat* y termina con el símbolo *FP*». Por ejemplo, una *FV*([ ]) está formada por una *FV*([*FP*]) seguida por una *FP*. La primera línea dice que *FV* con la lista de subcategorización *subcat* puede ser formada por un *Verbo* con la misma lista de subcategorización. Por ejemplo, una *FV*([*FN*]) puede ser formada por un *Verbo*([*FN*]). Un ejemplos de tal verbo es «tomar», por tanto «toma el oro» es una *FV*([ ]).

El segundo paso es cambiar la regla para *S* para decir que requiere de una frase verbal que tenga todos sus complementos y por tanto tenga la lista subcat [ ]. Esto significa que «Yo tomo el oro» es una frase legal, pero «Tú das» no lo es. La nueva regla

$$S \rightarrow FN(Sujeto)\ FV([\,])$$

Verbo	Subcat.	Ejemplo de frase verbal
Dar	[*FN*, *FP*] [*FN*, *FP*]	Dame el oro en 3 3 Dame el oro
Oler	[*FN*] [*Adjetivo*] [*FP*]	Huelo un *wumpus* Huele fatal Huele como un *wumpus*
Es	[*Adjetivo*] [*FP*] [*FN*]	Es apestoso Está en 2 2 Es un foso
Muerto Cree	[ ] [*S*]	Muerto Creo que el *wumpus* está muerto

Figura 22.12 Ejemplos de verbos con sus listas de subcategorización.

puede ser leída como «Una frase puede ser compuesta por una *FN* en el caso subjetivo, seguido por una *FV* que tenga una lista de subcat nula». La Figura 22.13 muestra un árbol sintáctico empleando esta gramática.

ADJUNTOS

El tercer paso es recordar que, además de los complementos, las frases verbales (y otras frases) pueden también actuar como **adjuntos**, que son frases que no son autorizadas por el verbo individual sino más bien pueden aparecer en cualquier frase verbal. Las frases que representan tiempo y espacio son adjuntas, porque casi cualquier acción o evento puede tener un tiempo o espacio. Por ejemplo, el adverbio «ahora» en «Yo huelo un *wumpus* ahora» y la *FP* «el martes» en «dame le oro el martes» son adjuntos. Aquí hay dos reglas que permiten adjuntos porposicionales y adverbiales en cualquier *FV*:

$$\begin{aligned} FV(subcat) \rightarrow\ & FV(subcat)\ FP \\ \mid\ & FV(subcat)\ Adverbio \end{aligned}$$

Figura 22.13 Árbol de análisis gramatical para «You give me the gold» (Dame el oro) mostrando la sucategorización del verbo y la frase verbal.

Capacidad generativa de las gramáticas aumentadas

ESQUEMA DE REGLAS

Cada regla aumentada es un **esquema de reglas**, que declara un conjunto de reglas, una para cada posible combinación de valores para los componentes aumentados. La capacidad generativa de las gramáticas aumentadas depende del número de combinaciones. Si hay un número finito, entonces la gramática aumentada es equivalente a una gramática independiente del contexto: el esquema de reglas podría ser reemplazado con reglas individuales independientes del contexto. Pero si hay un número infinito de valores, entonces las gramáticas aumentadas pueden representar lenguajes no independientes del contexto. Por ejemplo, el lenguaje dependiente del contexto $a^n b^n c^n$ puede ser representado como:

$$
\begin{array}{ll}
\mathrm{S}(n) \rightarrow A(n)B(n)C(n) & \\
A(1) \rightarrow \mathbf{a} & A(n+1) \rightarrow \mathbf{a}\ A(n) \\
B(1) \rightarrow \mathbf{a} & B(n+1) \rightarrow \mathbf{b}\ B(n) \\
C(1) \rightarrow \mathbf{a} & C(n+1) \rightarrow \mathbf{c}\ C(n)
\end{array}
$$

22.5 Interpretación semántica

Desde hace un buen rato, hemos examinado sólo el análisis sintáctico del lenguaje. En esta sección, nos volcaremos en la **semántica**: la extracción del *significado* de las declaraciones. En este capítulo estamos utilizando la lógica de primer orden como representación del lenguaje, por tanto la interpretación semántica es el proceso de asociar una expresión en LPO con una frase. Intuitivamente, el significado de la frase «el *wumpus*» es la bestia grande y peluda que representamos en lógica con el término lógico $Wumpus_1$, y el significado de «el *wumpus* está muerto» es la frase lógica $Muero(Wumpus_1)$. Esta sección hará esta intuición más precisa. Comenzaremos con ejemplos simples: una regla para describir posiciones en una rejilla:

$$FN \rightarrow Dígito\ Dígito$$

Aumentaremos la regla añadiendo a cada componente un argumento representativo de la semántica del componente. Obtenemos

$$FN([x,y]) \rightarrow Dígito(x)\ Dígito(y)$$

Esto significa que una cadena compuesta de un dígito con la semántica x seguida de otro dígito con la semántica y forma una *FN* con semántica $[x, y]$, que es nuestra notación para un cuadrado en la rejilla.

SEMÁNTICA COMPOSICIONAL

Destaquemos que la semántica de la *FN* completa está compuesta en su mayor parte de las semánticas de las partes componentes. Hemos visto esta idea antes en la **semántica composicional**: en lógica, el significado de $P \wedge Q$ se determina por el significado de P, Q y $\wedge$; en aritmética, el significado de $x + y$ se determina por el significado de x, y y $+$. La Figura 22.14 muestra cómo la notación DCG puede usarse para aumentar una gramática para expresiones aritméticas con semántica y la Figura 22.15 muestra el árbol sintáctico para $3 + (4 \div 2)$ de acuerdo con esta gramática. La raíz del árbol sintáctico es $Exp(5)$, una expresión cuya interpretación semántica es 5.

$$
\begin{array}{l}
Exp(x) \rightarrow Exp(x_1)\ Operador(op)\ Exp(x_2)\ \{x = \mathrm{A}plicar(op, x_1, x_2)\} \\
Exp(x) \rightarrow (Exp(x)) \\
Exp(x) \rightarrow Número(x) \\
Número(x) \rightarrow Dígito(x) \\
Número(x) \rightarrow Número(x_1)\ Dígito(x_2)\ Exp(x_2)\ \{x = 10 \text{ x } x_1 + x_2)\} \\
Dígito(x) \rightarrow x\ \{0 \leq x \leq 9\} \\
Operador(x) \rightarrow x\ \{x \in \{+, -, \div, \times\}\}
\end{array}
$$

Figura 22.14 Una gramática para expresiones aritméticas, aumentada con semántica. Cada variable x_i representa la semántica de un componente. Destaquemos el uso de la notación $\{test\}$ para definir predicados lógicos que deben ser satisfechos, pero que no son integrantes.

Figura 22.15 Árbol de análisis gramatical con las interpretaciones semánticas para la cadena «3 + (4 ÷ 2)».

La semántica de un fragmento en español

Ahora estamos preparados para escribir los aumentos semánticos para un fragmento de español. Comenzaremos determinando qué representaciones semánticas queremos asociar con qué frases. Usaremos como ejemplo la frase simple «Juan ama a María». La *FN* «Juan» debería de tener como su interpretación semántica el término lógico *Juan*, y la frase como un todo debería tener como interpretación la frase lógica *Ama*(*Juan*,*María*). Esto parece bastante claro. La parte complicada es la *FV* «ama a María». La interpretación semántica de esta frase no es ni un término lógico ni una frase completa lógica. Intuitivamente, «ama María» es una descripción que debería o no aplicarse a una persona particular. (En este caso se aplica a Juan.) Esto significa que «ama María» es un **predicado** que, cuando es combinado con un término que representa una persona (la persona amante), produce una frase lógica completa. Usando la notación λ (*véase* el Apartado 8.2), podemos representar «ama María» como el predicado

$$\lambda x\ Ama(x, Mary)$$

Ahora necesitamos una regla que diga «una *FN* con semántica *obj* seguida por una *FV* con semántica *rel* produce una frase cuya semántica es el resultado de aplicar *rel* a *obj*»:

$$S(rel(obj)) \rightarrow FN(obj)\ FV(obj)$$

La regla nos dice que la interpretación semántica de «Juan ama María» es

$$(\lambda x\ Ama(x, Mary))(John),$$

que es equivalente a *Ama*(*Juan*, *María*).

El resto de la semántica se sigue en una forma sencilla a partir de las elecciones que hemos hecho anteriormente. Debido a que las *FV* se representan como predicados, es una buena idea ser consistente y representar los verbos también como predicados. El verbo «ama» es representado como $\lambda x\ \lambda y\ Ama(x, y)$, el predicado que, cuando se da el argumento *María*, devuelve el predicado $\lambda x\ Ama(x, María)$.

La regla $FV \rightarrow Verbo\ FN$ se aplica al predicado que es la interpretación semántica del verbo al objeto que es la interpretación semántica de la *FN* para obtener la interpretación semántica de la *FV* completa. Terminaremos con la gramática que se muestra en la Figura 22.16 y el árbol sintáctico que se muestra en la Figura 22.17.

$S(rel(obj)) \rightarrow FN(obj)\ FV(rel)$
$FV(rel(obj)) \rightarrow Verbo(rel)\ FN(obj)$
$FN(obj) \rightarrow Nombre(obj)$

$Nombre(John) \rightarrow$ **John**
$Nombre(Mary) \rightarrow$ **Mary**
$Verbo(\lambda x\ \lambda y\ Ama(x, y)) \rightarrow$ **ama a**

Figura 22.16 Una gramática que puede derivar un árbol de análisis gramatical y una interpretación semántica como «John ama a Mary» (y otras tres oraciones). Cada categoría es aumentada con un único argumento representando la semántica.

Figura 22.17 Árbol de análisis gramatical con las interpretaciones semánticas para la cadena «John loves Mary» (John ama a Mary).

Tiempo y forma verbal

Ahora supongamos que queremos representar la diferencia entre «Juan ama a María» y «Juan amó a María». El inglés (castellano) usa los tiempos verbales (pasado, presente y futuro) para indicar el tiempo relativo de un evento. Una buena elección para representar el tiempo de los eventos es la notación del cálculo de eventos del Apartado 10.3. En la notación de eventos, nuestras dos frases tienen las siguientes interpretaciones:

$$e \in Ama(John, Mary) \wedge Durante(Ahora, e);$$
$$e \in Ama(John, Mary) \wedge Después(Ahora, e)$$

Esto sugiere que nuestras dos reglas léxicas para las palabras «ama» y «amó» deberían ser estas:

$$Verbo(\lambda x\ \lambda y\ e \in Ama(John, Mary) \wedge Durante(Ahora, e)) \rightarrow \textbf{ama};$$
$$Verbo(\lambda x\ \lambda y\ e \in Ama(John, Mary) \wedge Después(Ahora, e)) \rightarrow \textbf{amaba};$$

Excepto este cambio, cualquier otra cosa más de la gramática se mantiene igual, lo cual es una noticia alentadora; esto sugiere que estamos en la pista correcta si podemos añadir una complicación como el tiempo de los verbos (aunque tan sólo hemos rascado la superficie de una gramática completa para el tiempo y tiempo verbal). Con este éxito como puesta a punto (calentamiento), estamos ahora preparados para tratar un problema de representación mucho más complicado.

Cuantificación

Consideremos la frase «Cada agente huele un *wumpus*». La frase es realmente ambigua; el significado preferido es que los agentes podrían estar oliendo diferentes *wumpus*, pero un significado alternativo es que hay un único *wumpus* que todo el mundo huele[13]. Las dos interpretaciones se pueden representar como sigue:

$$\forall a\ a \in Agentes \Rightarrow$$
$$\quad \exists w\ w \in Wumpus \wedge \exists e\ e \in Huele(a, w) \wedge Durante(Ahora, e);$$
$$\exists w\ w \in Wumpus\ \forall a\ a \in Agentes \Rightarrow$$
$$\quad \exists e\ e \in Huele(a, w) \wedge Durante(Ahora, e).$$

Pospondremos el problema de la ambigüedad y por ahora analizaremos sólo la primera interpretación. Intentaremos analizarla composicionalmente, partiéndola en las componentes *FN* y *FV*:

Cada agente	$FN(\forall a\ a \in Agentes \Rightarrow F)$
huele un *wumpus*	$FV(\exists w\ w \in Wumpus \wedge$
	$\exists e\ (e \in Huele(a, w) \wedge Durante(Ahora, e))$

Ahora mismo hay dos dificultades. Primera, la semántica de una frase entera parece ser la semántica de la *FN*, con la semántica de la *FV* llenando la parte *F*. Esto significa que no podemos formar la semántica de la frase *rel*(*obj*). Podríamos hacer esto con *obj*(*rel*), que parece un poco extraño (por lo menos a primera vista). El segundo problema es que necesitamos obtener la variable *a* como un argumento de la relación *Huele*. En otras palabras, la semántica de una frase está formada mediante la conexión (*plugging*) de la semántica de la *FV* en el lugar correcto del argumento de la semántica de *FV*. Parece como si necesitásemos dos composiciones funcionales y promete ser más confuso aún. La complejidad tiene su origen en el hecho de que la estructura semántica es muy diferente de la estructura sintáctica.

FORMA INTERMEDIA

Para evitar esta confusión, muchos gramáticos modernos toman un rumbo diferente. Ellos definen una **forma intermedia** entre la sintaxis y la semántica. La forma intermedia tiene dos propiedades clave. Primera, es estructuralmente similar a la sintaxis de una frase

[13] Si esta interpretación parece incorrecta, consideremos «Cada protestante cree en un Dios justo».

y por tanto puede ser fácilmente construida mediante significados composicionales. Segunda, contiene la suficiente información para que sea traducida a una frase lógica de primer orden. Debido a que se encuentra entre las formas sintáctica y lógica, se llama **forma casi-lógica**[14]. En esta sección, usaremos una forma casi-lógica que incluye todo lo de la lógica de primer orden y es aumentada mediante expresiones lambda y una nueva construcción, que llamaremos **término cuantificado**. El término cuantificado que es la interpretación semántica de «cada agente» se escribe

FORMA CASI-LÓGICA

TÉRMINO CUANTIFICADO

$$[\forall a\ a \in Agentes]$$

Esta parece una sentencia lógica, pero se usa de la misma forma en la que es usado un término lógico. La interpretación de «Cada agente huele un *wumpus*» en forma casi-lógica es

$$\exists e\ (e \in Olfatea([\forall a\ a \in Agentes], [\exists w\ w \in Wumpus]) \wedge Durante(Ahora, e)$$

Para generar formas casi-lógicas, varias de nuestras reglas permanecen invariables. La regla para *S* todavía crea la semántica de *S* con *rel*(*obj*). Algunas reglas cambian; la regla léxica para «a» es

$$Artículo(\exists) \rightarrow \mathbf{a}$$

y la regla para combinar un artículo con un sustantivo es

$$FN([q\ x\ sem(x)]) \rightarrow Artículo(q)\ Sustantivo(sem)$$

Esta dice que la semántica de la *FN* es un término cuantificado, con un cuantificador especificado por el artículo, con la nueva variable *x*, y con una proposición formada mediante la aplicación de la semántica del sustantivo a la variable *x*. Las otras reglas de *FN* son similares. La Figura 22.18 muestra los tipos de semántica y formas ejemplo de cada categoría sintáctica bajo el enfoque de la forma casi-lógica. La Figura 22.19 muestra el árbol sintáctico de «cada agente huele un *wumpus*» usando este enfoque, y la Figura 22.20 muestra la gramática completa.

Ahora necesitamos convertir la forma casi-lógica en una lógica de primer orden real mediante la conversión de términos cuantificados en términos reales. Esto se hace mediante una simple regla: para cada término cuantificado $[q\ x\ P(x)]$ dentro de la forma casi-lógica *QLF*, reemplazar el término cuantificado con *x*, y reemplazar *QLF* con $q\ x\ P(x)\ op\ QLF$, donde *op* es $\Rightarrow$ cuando *q* es $\forall$ y es $\wedge$ cuando *q* es $\exists$ o $\exists!$ Por ejemplo, la frase «Cada perro tiene un día» tiene la forma casi-lógica:

$$\exists e\ (e \in Tiene([\forall d\ d \in Perros], [\exists a\ a \in Días], Ahora))$$

No especificamos cuál de los dos términos cuantificados será sacado primero, por tanto hay realmente dos posibles interpretaciones lógicas:

$$\forall d\ d \in Perros \Rightarrow \exists a\ a \in Días \wedge \exists e\ e \in Tiene(d, a, Ahora);$$
$$\exists\ a\ a \in Días \wedge \forall d\ d \in Perros \Rightarrow \exists e\ e \in Tiene(d, a, Ahora)$$

[14] Algunas formas casi-lógicas tienen una tercera propiedad que pueden sucintamente representar ambigüedades que podrían ser representadas sólo en forma lógica mediante una disyunción larga (*long*).

Categoría	Tipo Semántico	Ejemplo	Forma casi-lógica
S	*Oración*	Yo duermo.	$\exists e\ e \in Duerme(Narrador) \wedge Durante(Ahora, e)$
FN	objeto	un perro	$[\exists d\ Perro(d)]$
FF	$objeto^2 \rightarrow oración$	en [2, 2]	$\lambda x\ En(x, [2, 2])$
OraciónRel	$objeto \rightarrow oración$	eso me ve	$\lambda x\ \exists e\ e \in Ve(x, Narrador) \wedge Durante(Ahora, e)$
FV	$objeto^n \rightarrow oración$	me ve	$\lambda x\ \exists e\ e \in Ve(x, Narrador) \wedge Durante(Ahora, e)$
Adjetivo	$objeto \rightarrow oración$	mal cliente	$\lambda x\ MalCliente(x)$
Adverbio	$evento \rightarrow oración$	hoy	$\lambda e\ Durante(e, Hoy)$
Artículo	*cuantificador*	él	$\exists!$
Conjunción	$oración^2 \rightarrow oración$	y	$\lambda p, q\ (p \wedge q)$
Dígito	*objeto*	7	7
Sustantivo	$objeto \rightarrow oración$	*wumpus*	$\lambda x\ x \in Wumpuses$
Preposición	$objeto^2 \rightarrow oración$	en	$\lambda x\ \lambda y\ In(x, y)$
Pronombre	*objeto*	yo	*Narrador*
Verbo	$objeto^n \rightarrow oración$	come	$\lambda y\ \lambda x\ \exists e\ e \in Come(x, y) \wedge Durante(Ahora, e)$

Figura 22.18 Tabla que muestra el tipo de una forma casi lógica expresión para cada categoría sintáctica. La anotación $t \rightarrow r$ denota una función que toma un argumento de tipo t y devuelve un resultado de tipo r. Por ejemplo, el tipo semántico para *Preposición es* $objeto^2 \rightarrow oración$, que significa que la semántica de una preposición es una función que, cuando es aplicada a dos objetos lógicos, producirá una oración lógica.

Figura 22.19 Árbol de análisis gramatical para la oración «Todo agente huele un *wumpus*», mostrando tanto la interpretaciones semánticas como la estructura sintáctica.

$S(rel(obj)) \rightarrow FN(obj)\ FV(rel)$
$S(conj(sem_1, sem_2)) \rightarrow S(sem_1)\ Conjunción(conj)\ S(sem_2)$

$FN(sem) \rightarrow Pronombre(sem)$
$FN(sem) \rightarrow Nombre(sem)$
$FN([q\ x\ sem(x)]) \rightarrow Artículo(q)\ Nombre(sem)$
$FN([q\ x\ obj \wedge rel(x)]) \rightarrow FN([q\ x\ obj])\ FF(rel)$
$FN([q\ x\ obj \wedge rel(x)]) \rightarrow FN([q\ x\ obj])\ OraciónRel(rel)$
$FN([sem_1, sem_2]) \rightarrow Dígito(sem_1)\ Dígito(sem_2)$

$FV(sem) \rightarrow Verbo(sem)$
$FV(rel(obj)) \rightarrow FV(rel)\ FN(obj)$
$FV(sem_1(sem_2)) \rightarrow FV(sem_1)\ Adjetivo(sem_2)$
$FV(sem_1(sem_2)) \rightarrow FV(sem_1)\ PP(sem_2)$

$OraciónRel(sem) \rightarrow$ **ese** $FV(sem)$

$FF(\lambda x\ rel(x, obj)) \rightarrow Preposición(rel)\ FN(obj)$

Figura 22.20 Una gramática con semántica en forma casi-lógica.

La primera dice que cada perro tiene su propio día, mientras que la segunda dice que hay un día especial que todos los perros comparten. Elegir entre ellas es un trabajo para deshacer la ambigüedad. A menudo, el orden de izquierda a derecha de los términos cuantificados empareja el orden de izquierda a derecha de los cuantificadores, pero otros factores entran en juego. La ventaja de la forma casi-lógica es que sucintamente representa todas las posibilidades. La desventaja es que no nos ayuda a elegir entre ellas; para ello necesitamos la potencia plena de la desambiguación empleando toda fuente de evidencias.

Interpretación pragmática

Hemos mostrado cómo un agente puede percibir una cadena de palabras empleando una gramática para derivar un conjunto de posibles interpretaciones semánticas. Ahora hablaremos del problema de la completitud de la interpretación mediante la adición de información dependiente del contexto sobre la situación actual de cada interpretación candidata.

REFERENTES

La necesidad más obvia de información pragmática es en la resolución del significado de **referentes**, que son frases que hacen referencia directamente a la situación actual. Por ejemplo, en la frase «yo estoy en Boston hoy», la interpretación de los referentes «yo» y «hoy» depende de quién declare la frase. Presentamos referentes mediante «constantes» (tales como *Hablante*) que realmente son **locuaces** (esto significa, que dependen de la situación). El oyente que percibe un acto de habla debería también percibir qué hablante es y usar esta información para resolver el referente. Por ejemplo, el oyente debería saber $T((Hablante = Agente_B), Ahora)$.

Un comando tal como «ve a 2 2» implícitamente se refiere al oyente. Más aún, nuestra gramática para *S* cubre sólo frases declarativas. Podemos fácilmente extenderlas para cubrir órdenes[15].

Una orden se puede formar a partir de una *FV*, donde el sujeto es implícitamente el oyente. Necesitamos distinguir las órdenes de las afirmaciones, así podemos alterar las reglas de *S* para incluir el tipo de acto de habla como parte de la forma casi-lógica:

$$S(Sentencia(Hablante, rel(obj))) \rightarrow FN(obj)\ FV(rel)$$
$$S(Orden(Hablante, rel(Oyente))) \rightarrow FV(rel)$$

Por tanto, la forma casi-lógica para «Ve a 2 2» es[16]

$$Orden(\exists e\ e \in Ir(Oyente, [2, 2]))$$

Generación de lenguajes con DCGs

Desde hace rato nos hemos concentrado en *análisis sintáctico* del lenguaje, no en la generación del mismo. La generación es un tema de similar riqueza. Elegir la declaración correcta para expresar una proposición involucra a muchas de las mismas alternativas que la declaración del análisis gramatical emplea.

Recordemos que una DCG es un sistema de programación lógico que especifica restricciones entre una cadena y el análisis gramatical de una cadena. Conocemos que una definición de la programación lógica del predicado *Append* se puede usar tanto para decirnos que en *Append*([1, 2], [3], *x*) tenemos $x = [1, 2, 3]$ y que para enumerar los valores de *x* e *y* tenemos que hacer que *Append*(*x*, *y*, [1, 2, 3]) sea verdad. De la misma manera, podemos escribir una definición de *S* que puede ser usado de dos formas: para el análisis gramatical, preguntamos *S*(*sem*, [*Juan*, *Ama*, *María*]) y obtenemos *sem* = *Ama*(*Juan*, *María*); para generar, preguntamos *S*(*Ama*(*Juan*, *María*), *words*) y obtenemos *words* = [*Juan*, *Ama*, *María*]. Podemos también comprobar una gramática preguntando *S*(*sem*, *words*) y obtenemos como respuesta un raudal (flujo) de pares [*sem*, *words*] que son generados por la gramática.

Este enfoque funciona para las gramáticas simples de este capítulo, pero puede haber dificultades en llevarlo a gramáticas más grandes. La estrategia de búsqueda empleada por el motor de inferencia lógica es importante; las estrategias de búsqueda en profundidad pueden avocar en bucles infinitos. Se deben tomar algunas precauciones en los detalles precisos de la forma semántica. Podría ser que una determinada gramática no tuviera manera de expresar la forma lógica $X \wedge Y$ para algunos valores de *X* e *Y*, pero

---

[15] Para implementar un agente de comunicaciones completo necesitamos también una gramática de preguntas. Las preguntas están más allá del alcance de este libro porque imponen una **dependencia de gran distancia** entre componentes. Por ejemplo, en «Whom did the agent tell you to give the gold to?» la palabra final «to» debería ser analizada sintácticamente como una *FP* con la falta de un *FN*; la falta de *FN* se autoriza por la primera palabra de la frase, «who». Se emplea un sistema completo de aumentos para asegurar que el *FN* ausente se empareja con las palabras autorizadas.

[16] Destacar que la forma casi-lógica para una orden no incluye el tiempo del evento (por ejemplo, *During*(*Now*, *e*)). Esto es debido a que «go» es realmente la versión sin tiempo (no conjugada) de la palabra, no la versión en tiempo presente. No se puede comunicar la diferencia con «go», pero observen que la forma correcta de una orden es «Be good!» (usando la forma no conjugada «be»), no «Are good!». Para asegurar que se usa la conjugación correcta, podríamos aumentar las *FV* con un argumento de conjugación y escribir *VP*(*rel*, *untensed*) en el lado derecho de la regla de la orden.

pueda expresar $Y \wedge X$; esto sugiere que necesitamos una forma de canonicalizar formas semánticas, o bien que necesitamos ampliar la rutina de unificación de modo que $X \wedge Y$ se pueda unificar con $Y \wedge X$.

Los trabajos formales en generación tienden a usar modelos de generación más complejos que son distintos del análisis gramatical y ofrecen más control sobre qué componentes semánticos se expresan. La gramática sistémica es un enfoque que hace fácil poner énfasis en las partes más importantes de la forma semántica.

22.6 Ambigüedad y desambigüedad

En algunos casos, los oyentes son conscientemente avisados de la ambigüedad en una declaración. Aquí hay varios ejemplos tomados de los titulares de los periódicos:

Squad helps dog bite victim.
Helicopter powered by human flies.
Once sagging cloth diaper industry saved by full dumps.
Portable toilet bombed; police have nothing to go no.
British left waffles on Falkland Islands.
Teacher strikes idle kids.
Milk drinkers are turning to powder.
Drunk gets nine months in violin case[17].

Pero la mayoría del tiempo el lenguaje que oímos parece no ambiguo. Así, cuando los investigadores comenzaron a usar por primera vez computadores para analizar el lenguaje en los años 60, quedaron bastante sorprendidos al aprender que *casi toda declaración es altamente ambigua, incluso cuando las interpretaciones alternativas no deberían ser apreciables a un orador nativo*. Un sistema con una amplia gramática y léxico podría encontrar miles de interpretaciones para una frase perfectamente ordinaria. Consideremos «the batter hit the ball», que parece tener una interpretación no ambigua en la cual el jugador de baseball golpea una bola. Pero obtenemos una interpretación diferente si la frase previa es «The mad scientist unleashed a tidal wave of cake mix towards the ballroom». Este ejemplo se basa en una **ambigüedad léxica**, en la cual una palabra tiene más de un significado. La ambigüedad léxica es bastante común; «back» puede ser un adverbio (*go back*), un adjetivo (*back door*), un sustantivo (*the back of the room*) o un verbo (back up your files). «Jack» puede ser un nombre, un sustantivo (un juego de cartas, una pieza de juego de metal con seis puntas, una bandera náutica, un pez, un asno macho, un bolsillo, o un dispositivo para levantar objetos pesados), o un verbo (*to jack up a car*, cazar con una linterna, o golpear una bola con dureza).

AMBIGÜEDAD LÉXICA

AMBIGÜEDAD SINTÁCTICA

La **ambigüedad sintáctica** (también conocida como **ambigüedad estructural**) puede ocurrir con o sin ambigüedad léxica. Por ejemplo, la cadena «I smelled a wumpus in 2,2» tiene dos análisis: uno donde la frase preposicional «in 2,2» modifica un sustantivo y otra donde modifica el verbo. La ambigüedad sintáctica avoca a una **ambigüedad semántica**,

AMBIGÜEDAD SEMÁNTICA

[17] *Nota del traductor:* Frases del original en inglés que debido a su ambigüedad no tienen traducción posible en español. Como ejemplo en español consideremos la frase «Perdí los gemelos en el parque», hace referencia a perder los gemelos (*niños*) en el parque o los gemelos (*broches*) de la camisa.

porque un análisis significa que el *wumpus* está en 2,2 y el otro significa que un tufo está en 2,2. En este caso, quedarse con la interpretación incorrecta podría ser un error mortal.

La ambigüedad semántica puede aparecer incluso en frases sin ambigüedad léxica o sintáctica. Por ejemplo la frase nominal «cat person» puede ser alguien a quien le gusten los gatos o el líder de la película «*Attack of the Cat People*». Una «coast road» puede ser una carretera que siga la costa o alguien que se dirige hacia ella.

Finalmente. Hay una ambigüedad entre significados literales y figurados. Las figuras del habla son importantes en poesía, pero son sorprendentemente comunes también en el habla cotidiana. Una **metonimia** es una figura del habla en la cual se emplea un objeto para sustituir a otro objeto. Cuando oímos «Chrysler anunció otro modelo», no lo interpretamos como que la compañía puede hablar; más bien entendemos que un portavoz representando la compañía hizo el anuncio. La metonimia es común y es a menudo interpretada inconscientemente por los oyentes humanos. Desafortunadamente, nuestra gramática tal y como es escrita no es tan fácil. Para manejar la semántica de la metonimia adecuadamente, necesitamos introducir un nuevo nivel completo de ambigüedad. Lo realizamos introduciendo *dos* objetos para la interpretación semántica de cada frase en la frase en la oración: uno para el objeto al que la frase literalmente hace referencia (Chrysler) y otro para la referencia metonímica (el portavoz). Entonces podemos decir que hay una relación entre los dos. En nuestra gramática actual, «Chrysler anunció» obtiene la interpretación como

METONIMIA

$$\exists x, e\ x = Chrysler \wedge e \in Anuncios(x) \wedge Después(Ahora, e)$$

Necesitamos cambiarlo a

$$\exists\ m, x, e\ x = Chrysler \wedge e \in Anuncios(m) \wedge Después(Ahora, e) \wedge Metonimia(m, x)$$

Esta dice que hay una entidad x que es igual a Chrysler, y otra entidad m que hizo el anuncio, y aquella que dice que las dos están en una relación metonímica. El siguiente paso es definir qué clases de relaciones metonímicas pueden darse. El caso más simple es cuando no hay metonimia después de todo (el objeto literal x y el objeto metonímico m son idénticos):

$$\forall m, x\ (m = x) \Rightarrow Metonimia(m, x)$$

Para el ejemplo de Chrysler, una generalización razonable es que se puede usar una organización para mantener un representante de dicha organización.

$$\forall m, x\ \ x \in Organizaciones \wedge Representante(m, x) \Rightarrow Metonimia(m, x)$$

Otras metonimias incluyen el autor para el trabajo (Yo leo *Shakespeare*) o más genéricamente el fabricante del producto (Yo conduzco un *Honda*) y la parte para el conjunto (El Calcetín Rojo necesita un *brazo* fuerte). Algunos ejemplos de metonimia, tales como «El *sándwich de jamón* en la Tabla 4 quiere otra cerveza», son más novedosos y son interpretados según la situación.

Las reglas que hemos esbozado aquí nos permiten construir una explicación para «Chrysler anunció un nuevo modelo», pero la explicación no sigue una deducción lógica. Necesitamos usar el razonamiento probabilístico o no monotónico para aparecer con explicaciones candidatas.

METÁFORA

Una **metáfora** es una figura del habla en la cual se emplea una frase con un significado literal para sugerir un significado diferente mediante una analogía. La mayoría de la gente piensa en las metáforas como una herramienta empleada por los poetas que no toman un gran papel en el texto diario. Sin embargo, una cierta cantidad de metáforas básicas son tan comunes que no las reconocemos como tales. Una metáfora de dicho tipo es la idea en la que «más es subir» (*more is up*). Esta metáfora nos permite decir que los precios han crecido, escalado, o subir como un cohete, que la temperatura ha bajado o caído, que una confidencia ha caído en picado o que una celebridad popular ha saltado o ascendido.

Hay dos maneras de acometer metáforas como esta. Una es recopilar todo el conocimiento de la metáfora en el léxico, para añadir nuevos significados de las palabras «crecido», «caer», «escalar» y demás, que las describe como manejo de cantidades de cualquier escala en vez de tan sólo altitud. Este enfoque es suficiente para gran cantidad de aplicaciones, pero no capta el carácter generativo de la metáfora que permite a los humanos usar nuevas instancias tales como «nosedive» o «blasting through the roof» sin sentimiento de no comprender. El segundo enfoque es incluir conocimiento explícito de las metáforas comunes y usarlas para interpretar nuevos usos a medida que son leídos. Por ejemplo, supongamos que el sistema conoce la metáfora «more is up». Esto es, conoce que dichas expresiones lógicas que se refieren a un punto en una escala vertical se pueden interpretar como pertenecientes a puntos correspondientes a una escala de cantidad. Entonces la expresión «las ventas son altas» debería tener una interpretación literal a través de las líneas de *Altitud* (*ventas*, *altas*), que debería ser interpretado metafóricamente como *Cantidad* (*ventas*, *muchas*).

Desambiguación

Como dijimos antes, *la desambiguación es un problema de diagnosis*. La intención del hablante para comunicar es una causa no observada de las palabras en la declaración, y el trabajo del oyente es trabajar hacia atrás a partir de las palabras y a partir del conocimiento de la situación para recuperar lo más similarmente la intención del hablante

$$\underset{intención}{\operatorname{argmax}}\ Probabilidad(intención \mid palabras, situación),$$

donde la probabilidad puede ser bien una probabilidad o cualquier medida numérica de preferencia. Se necesita algún orden de preferencia debido a que la regla de interpretación sintáctica y semántica en sí misma no puede identificar a una única interpretación correcta de una frase u oración. Por tanto dividimos el trabajo: la interpretación sintáctica y semántica es responsable de la enumeración de un conjunto de interpretaciones candidatas, y el proceso de desambiguación elige la mejor.

Destaquemos que hemos hablado sobre la intención del acto del habla, no la proposición actual que el hablante está proclamando. Por ejemplo, después de oír a un político decir, «yo no soy un delincuente», podríamos asignarle una probabilidad de sólo el 50 por ciento a la proposición que dice que el político no es un criminal, y un 99,999

por ciento a la proposición que dice que el hablante no está enganchado al shepherd's staff. Aun así, debemos asignar una probabilidad más alta a la interpretación

$$Aserción(Hablante, \neg (Hablante \in Criminales))$$

porque es más fácil pensarlo que decirlo.

ASOCIACIÓN POR LA DERECHA

Consideremos de nuevo el ejemplo ambiguo «yo olfateé un *wumpus* en 2,2». Una preferencia heurística es la regla de la **asociación por la derecha**, que dice que cuando sea el momento de decidir dónde colocamos la *FP* «en 2,2» en el árbol de análisis gramatical, deberíamos preferir asociarla al componente existente de más hacia la derecha, que en este caso es la *FN* «un *wumpus*». De acuerdo, esto es sólo una heurística; para la oración «yo olfateé un *wumpus* con mi nariz», la heurística debería ser sopesada por el hecho en el que rechazamos la *FN* «un *wumpus* con mi nariz».

La desambiguación se hace posible mediante la combinación de la evidencia, usando todas las técnicas para la representación del conocimiento y el razonamiento no certero que hemos visto a lo largo de este libro.

Podemos partir el conocimiento en cuatro modelos:

1. El **modelo del mundo**: la probabilidad de que una proposición ocurra en el mundo.
2. El **modelo mental**: la probabilidad de que el hablante forme la intención de comunicar un cierto hecho al oyente, tal como ocurrió. Este enfoque combina modelos de lo que el hablante cree y lo que el hablante cree que el oyente cree, etcétera.
3. El **modelo del lenguaje**: la probabilidad de que una cierta cadena de palabras sea elegida, dado que el hablante tiene la intención de comunicar un cierto hecho. Los modelos CFG y DCG planteados en este capítulo tienen un modelo Booleano de probabilidad: bien una cadena puede tener una cierta interpretación o no tenerla. En el próximo capítulo veremos una versión probabilística de CFG que proporciona un modelo de lenguaje más enriquecido para la desambiguación.
4. El **modelo acústico**: la probabilidad de que una secuencia particular de sonido sea generada, dada porque el hablante ha elegido una cadena de palabras determinada. El Apartado 15.6 cubría el reconocimiento del habla.

22.7 Comprensión del discurso

DISCURSO

Un **discurso** es cualquier cadena del lenguaje —usualmente una que tiene más de una frase de longitud. Libros de texto, novelas, informes metereológicos y conversaciones son discursos. Desde hace rato hemos ignorado ampliamente los problemas del discurso, y hemos preferido diseccionar el lenguaje en oraciones individuales que pueden ser estudiadas *in vitro*. Esta sección estudia las oraciones en su hábitat nativo. Buscaremos dos subproblemas particulares: la resolución por referencia y la coherencia.

Resolución por referencia

RESOLUCIÓN POR REFERENCIA

La **resolución por referencia** es la interpretación de un pronombre o una frase nominativa definitiva que hace referencia a un objeto en el mundo[18]. La resolución se basa en el conocimiento del mundo en las partes previas del discurso. Consideremos el pasaje

«Juan paró al camarero. Él pidió un sándwich de jamón».

Para entender que «él» en la segunda oración se refiere a Juan, necesitamos haber entendido que en la primera oración menciona dos personas y que Juan está ejerciendo el papel de cliente y por ello es capaz de pedir mientras que el camarero no lo es. Usualmente, la resolución por referencia es un problema de seleccionar una referencia de una lista de candidatas, pero a veces implica la creación de nuevas candidatas. Consideremos la siguiente oración:

«Después de que Juan se declarase a Marsha, ellos encontraron un predicador y se casaron. Para la luna de miel ellos se fueron a Hawai».

Aquí, la frase nominativa «la luna de miel» se refiere a algo que es aludido implícitamente sólo por el verbo «casados». El pronombre «ellos» se refiere a un grupo que no fue explícitamente mencionado antes: Juan y Marsha (pero *no* el predicador).

Elegir la mejor referencia es un proceso de desambiguación que implica la combinación de una variedad de información sintética, semántica y pragmática. Algunas pistas están en la forma de las restricciones. Por ejemplo, los pronombres deben concordar en género y número con sus precedentes: «él» puede hacer referencia a Juan, pero no a Marsha; «ellos» se pueden referir a un grupo, pero no a una única persona. Los pronombres también deben obedecer restricciones sintácticas por reflexividad. Por ejemplo, en «él lo vio en el espejo» los dos pronombres deben referirse a personas diferentes, mientras que en «él se vio a sí mismo» se deben referir a la misma persona. Hay también restricciones para la consistencia semántica. En «él se lo comió», el pronombre «él» debe referirse a algo que come y «lo» a algo que puede ser comido.

Algunas pistas son preferencias que no siempre se mantienen. Por ejemplo, cuando oraciones adyacentes tienen una estructura paralela, es preferible para referencia pronominal seguir dicha estructura. Por tanto, en

Marsha voló a San Francisco desde Nueva York. Juan voló allí desde Boston.

preferimos para «allí» referirnos a San Francisco porque sigue las mismas reglas sintácticas. Ausentes a una estructura paralela, hay una preferencia para los sujetos por encima de los objetos como precedentes. Así, en

Marsha le dio a Sally la asignación de las tareas de casa. Entonces ella se fue.

«Marsha», el sujeto de la primera frase, es el antecedente preferido para «ella». Otra preferencia es para la entidad que ha sido discutida más prominentemente. Consideremos aisladamente el par de oraciones

Dana dejó caer el vaso sobre el plato. Éste se rompió.

---

[18] En lingüística, referenciar algo que ya ha sido introducido se llama referencia **anafórica**. Referenciar algo que todavía tiene que ser introducido se llama referencia **catafórica**, como en el caso del pronombre «él» en «cuando él ganó su primer torneo, los Tigres tenían 20».

plantea un problema: no está claro si bien el vaso o el plato es el referente de «Éste». Pero en un contexto más grande la ambigüedad es resuelta:

> Dana tranquilamente encontró el vaso azul. El vaso había sido un regalo de un amigo íntimo. Desafortunadamente, un día mientras estaba sentada en la mesa, Dana dejó caer el vaso sobre el plato. Éste se rompió.

Aquí, el vaso es el foco de atención y por ello es el referente preferido.

Han sido diseñados una variedad de algoritmos de resolución por referencia. Uno de los primeros (Hobbs, 1998) es destacable porque experimenta un grado de verificación estadística que era inusual para la época. Usando tres géneros diferentes de texto, Hobbs informó de una eficacia del 92 por ciento. Esto asumía que un análisis gramatical correcto era generado por un analizador gramatical; sin tener uno disponible, Hobbs construyó los analizadores gramaticales a mano. El algoritmo de Hobbs funcionó como investigación: éste buscaba oraciones comenzando desde la oración actual y yendo hacia atrás. Esta técnica asegura que los candidatos más recientes serán considerados en primer lugar. Dentro de una frase busca primero en amplitud, de izquierda a derecha. Esto asegura que los sujetos serán considerados antes que los objetos. El algoritmo elige el primer candidado que satisface las restricciones perfiladas.

La estructura de un discurso coherente

Abra 10 páginas al azar de este libro, y copie la primera oración de cada página. El resultado está comprometido por ser incoherente. Similarmente, si elige un pasaje de 10 oraciones coherentes y permuta las oraciones, el resultado es incoherente. Esto demuestra que las oraciones en el discurso del lenguaje natural son bastante distintas de las oraciones en lógica. En lógica REFERIMOS las oraciones A, B y C como base de conocimiento, en cualquier orden, terminamos con la conjunción $A \wedge B \wedge C$. En el lenguaje natural, el orden de las oraciones importa; consideremos la diferencia entre «Avanza dos bloques. Gira a la derecha» y «Gira a la derecha. Avanza dos bloques».

Un discurso tiene una estructura superior al nivel de una oración. Podemos examinar esta estructura con la ayuda de una gramática del discurso:

$$Segmento(x) \rightarrow S(x)$$
$$Segmento(RelaciónCoherencia(x, y)) \rightarrow Segmento(x)\ Segmento(y).$$

Esta gramática dice que un discurso está compuesto de segmentos, donde cada segmento es bien una oración o un grupo de oraciones y donde los segmentos se unen mediante **relaciones de coherencia**. En el texto «Avanza dos bloques. Gira a la derecha», la relación de coherencia es que la primera oración hace posible la segunda: el oyente debería girar a la derecha después de viajar dos bloques. Diferentes investigadores han propuesto diferentes inventarios de relaciones de coherencia; la Figura 22.21 enumera un conjunto representativo. Consideremos la siguiente historia:

RELACIONES DE COHERENCIA

(1) Algo divertido ocurrió ayer.
(2) Juan fue a un restaurante de lujo.
(3) El pidió pato.

- **Habilitar** o **causar**: S_1 proporciona un cambio de estado (que puede ser implícito) que causa o habilita S_2. Ejemplo: «yo salí. Conduje al colegio». (Salir habilita el sentido implícito de conducir un coche).
- **Explicación**: La inversa de la habilitación: S_2 causa o habilita a S_1 y por tanto es una explicación para ella. Ejemplo: «Era tarde para la escuela. Me dormí».
- **Figura subyacente**: S_1 describe una asignación para S_2. Ejemplo: «Era una noche oscura y tormentosa. Resto de la historia».
- **Evaluación**: A partir de S_2 se infiere que S_1 es parte del plan del hablante para ejecutar el segmento como un acto del habla. Ejemplo: «Algo divertido ocurrió. Resto de la historia».
- **Ejemplificación**: S_2 es un ejemplo del principio general en S_1. Ejemplo: «Este algoritmo invierte una lista. La entrada $[A, B, C]$ se transforma en $[C, B, A]$.
- **Generalización**: S_1 es un ejemplo del principio general en S_2. Ejemplo: «$[A, B, C]$ se mapea a $[C, B, A]$. En general, el algoritmo invierte una lista».
- **Expectativa incumplida**: Se infiere $\neg P$ a partir de S_2, negando la inferencia normal de P a partir de S_1. Ejemplo: «Este artículo es flojo. Por otro lado, es interesante».

Figura 22.21 Una lista de relaciones de coherencia, tomadas de Hobbs (1990). Cada relación se mantiene entre dos segmentos adyacentes de texto, S_1 y S_2.

(4) La factura alcanzó los 50 dólares.
(5) Juan tuvo un trauma cuando se percató de que no tenía dinero.
(6) Él había dejado su billetera en casa.
(7) El camarero dijo que podía pagar más tarde.
(8) Él estaba muy avergonzado por su olvido.

Aquí, la oración (1) mantiene la relación de *Evaluación* en el resto del discurso; (1) es el metacomentario del hablante en el discurso. La oración (2) habilita la (3), y juntas el par (2-3) causan la (4), con el estado intermedio implícito en el que Juan comió el pato. Ahora (2-4) sirven como base para el resto del discurso. La oración (6) es una explicación de la (5) y (5-6) permiten la (7). Hay que darse cuenta de que es una *Habilitación* y no una *Causa*, porque el camarero podría haber tenido una reacción diferente. Juntas, las (5-7) causan la (8). El Ejercicio 22.13 pide dibujar un árbol de análisis gramatical para este discurso.

Las relaciones de coherencia sirven para unir un discurso. Éstas guían al hablante en decidir qué decir y qué dejar implícito, y éstas guían al oyente en la recuperación de la intención del hablante. Las relaciones de coherencia pueden servir como un filtro en la ambigüedad de las oraciones: individualmente, las oraciones podrían ser ambiguas, pero la mayoría de estas interpretaciones ambiguas no se ajustan a la vez en un discurso coherente.

Desde hace tiempo hemos buscado una resolución de la referencia y una estructura del discurso de forma separada. Pero las dos están realmente entrelazadas. La teoría de Grosz y Sidner (1986), por ejemplo, explica que dónde se dirige la atención del hablante y del oyente durante un discurso. Su teoría incluye una pila descendente de los contextos **centrados en los espacios**. Ciertas declaraciones causan que el foco de atención se desplace mediante la apilación o desapilación de elementos de la pila. Por ejemplo, en

CENTRADOS EN LOS ESPACIOS

la historia del restaurante, la oración «Juan fue a un restaurante de lujo» apila un nuevo foco en la pila. Dentro de dicho foco, el hablante puede usar una *FN* definitiva para referirse «al camarero» (en vez de «un camarero»). Si la historia continuaba con «Juan fue a casa», entonces el espacio de atención debería ser desapilado de la pila y el discurso no podría nunca más referirse al camarero con «el camarero» o «él».

22.8 Inducción gramatical

INDUCCIÓN GRAMATICAL

La **inducción gramatical** es la tarea de aprender una gramática a partir de datos. Obviamente es una tarea a intentar, dado que está probado que es bastante difícil construir una gramática a mano y que billones de declaraciones ejemplo están disponibles gratuitamente en internet. Es una tarea difícil porque el espacio de las posibles gramáticas es infinito y porque verificar que una gramática dada genera un conjunto de oraciones es caro computacionalmente.

Un modelo interesante es el sistema SEQUITUR (Nevill-Manning y Witten, 1997). No requiere entradas excepto un texto sencillo (el cual no necesita ser predividido en oraciones). Éste produce una gramática de una forma muy especializada: una gramática que genera sólo una única cadena, particularmente, el texto original. Otra manera de verlo es que SEQUITUR aprende la suficiente gramática para analizar gramaticalmente el texto. Aquí está entrecorchetado lo que él descubre para una oración dentro de un texto más largo de historias de noticias:

[La mayoría de las labores][sentimiento[[debería aún][favor de]abolición]][[de[la Casa]][de lores]]

Ha elegido correctamente los componentes tales como la *FP* «de la casa de los lores», aunque también va contra el análisis tradicional, por ejemplo, agrupar «el» con el verbo precedente más que con el siguiente sustantivo.

SEQUITUR se basa en la idea de que una buena gramática es una gramática compacta. En particular, ella impone las dos siguientes restricciones: (1) Ningún par de símbolos adyacentes debería aparecer más de una vez en la gramática. Si el par de símbolos $A\ B$ aparece en la parte derecha de varias reglas, entonces deberíamos reemplazar el par con un nuevo no terminal que llamaremos C y añadir la regla $C \rightarrow A\ B$. (2) Cada regla debe ser usada al menos dos veces. Si un no terminal C aparece solamente una vez en la gramática, entonces deberemos eliminar la regla para C y reemplazar su único uso con la regla del lado derecho. Estas dos restricciones se aplican en una búsqueda voraz que examina el texto de entrada de izquierda a derecha, incrementalmente construyendo una gramática a medida que avanza, e imponiendo las restricciones tan pronto como es posible. La Figura 22.22 muestra el algoritmo en funcionamiento sobre el texto de entrada «*abcdbcabcd*». El algoritmo recupera una gramática compacta óptima para el texto.

En el siguiente capítulo, veremos otros algoritmos de inducción gramatical que funcionan con gramáticas independientes del contexto probabilísticas. Pero ahora volvemos al problema de aprender una gramática que es aumentada con semántica. Ya que una gramática aumentada es un programa lógico clausal de Horn, son adecuadas las técnicas de programación lógica inductiva. CHILL (Zelle y Mooney, 1996) es un programa de pro-

	Entrada	Gramática	Comentarios
1	*a*	$S \rightarrow a$	
2	*ab*	$S \rightarrow ab$	
3	*abc*	$S \rightarrow abc$	
4	*abcd*	$S \rightarrow abcd$	
5	*abcdb*	$S \rightarrow abcdb$	
6	*abcdbc*	$S \rightarrow abcdbc$ $S \rightarrow aAdA$; $A \rightarrow bc$	*bc* se repite
7	*abcdbca*	$S \rightarrow aAdAa$; $A \rightarrow bc$	
8	*abcdbcab*	$S \rightarrow aAdAab$; $A \rightarrow bc$	
9	*abcdbcabc*	$S \rightarrow aAdAabc$; $A \rightarrow bc$ $S \rightarrow aAdAaA$; $A \rightarrow bc$ $S \rightarrow BdAB$; $A \rightarrow bc$; $B \rightarrow aA$	*bc* se repite *aA* se repite
10	*abcdbcabcd*	$S \rightarrow BdABd$; $A \rightarrow bc$; $B \rightarrow aA$ $S \rightarrow CAC$; $A \rightarrow bc$; $B \rightarrow aA$; $C \rightarrow Bd$ $S \rightarrow \mathrm{C}AC$; $A \rightarrow bc$; $\mathrm{C} \rightarrow aAd$	*Bd* se repite *B* sólo una vez

Figura 22.22 Una traza de SEQUITUR introduciendo una gramática para el texto de entrada «*abcdbcabcd*». Comenzamos con una regla para *S* y añadimos cada símbolo al final de esta regla en orden. Tras añadir el sexto símbolo, tenemos la primera ocurrencia de un par repetido: *bc*. Por tanto reemplazamos ambas ocurrencias de *bc* con el nuevo no terminal *A* y añadimos la regla $A \rightarrow bc$. Después de añadir tres símbolos más, el noveno provoca otra repetición del *bc*, por tanto de nuevo lo reemplazamos con *A*. Esto nos conduce a dos ocurrencias de *aA*, por tanto las reemplazamos con un nuevo no terminal *B*. Después de añadir el décimo y último símbolo terminal obtenemos las dos ocurrencias de *Bd*, por tanto las reemplazamos con el nuevo no terminal *C*. Pero ahora *B* aparece sólo una vez, en el lado derecho de la regla *C*, por tanto reemplazamos *B* con su expansión, *aA*.

gramación lógica inductiva (ILP) que aprende una gramática de un analizador gramatical especializado para dicha gramática a partir de ejemplos. El dominio objetivo son consultas en la base de datos en lenguaje natural. Los ejemplos de entrenamiento consisten en pares de cadenas de palabras y sus correspondientes preguntas, por ejemplo:

¿Cuál es la capital del estado con la población más grande?
$Respuesta\ (c, Capital(s, c) \wedge Másgrande(p, Estado(s) \wedge Población(s, p)))$

La tarea de Chill es aprender un predicado *AnálisisGramatical*(*palabras*, *pregunta*) que sea consistente con los ejemplos y, de manera optimista, generalice de forma adecuada los otros ejemplos. La aplicación de ILP directamente en el aprendizaje de este predicado acaba en un pobre rendimiento: el analizador gramatical inducido tiene sólo sobre el 20 por ciento de precisión. Afortunadamente, los aprendices ILP pueden mejorar mediante

la adición de conocimientos. En este caso la mayoría del predicado *AnálisisGramatical* estaba definido como un programa lógico, y la tarea de CHILL fue reducida a la inducción de las reglas de control que guían al analizador gramatical para seleccionar un análisis gramatical sobre otros. Con este conocimiento adicional subyacente, CHILL alcanza una precisión del 70 al 80 por ciento sobre varias tareas de interrogación de bases de datos.

22.9 Resumen

La comprensión del lenguaje natural es uno de los subcampos más importantes de la IA. Se perfila sobre ideas de la filosofía lingüística, así como sobre técnicas de representación y razonamiento del conocimiento lógico y probabilístico. Al contrario que otras áreas de la IA, la comprensión del lenguaje natural necesita una investigación empírica del comportamiento actual humano, que se vuelve cada vez más complejo e interesante.

- Los agentes envían señales a los demás para alcanzar cierto propósito: para informar, para avisar, para solicitar ayuda, para compartir conocimiento, o para prometer algo. Enviar una señal en este medio se llama un **acto de habla**. Por último, todos los actos de habla son un intento de conseguir que otro agente que crea algo o haga algo.
- El lenguaje está formado de **signos** convencionales que son un convenio de significados. Muchos animales usan signos en este sentido. Los humanos parecen ser los únicos animales que usan la **gramática** para producir una variedad no limitada de mensajes estructurados.
- La comunicación implica tres pasos con el hablante: la intención de convenir una idea, la generación mental de las palabras, y su síntesis física. El oyente entonces tiene cuatro pasos: percepción, análisis, desambiguación, e incorporación del significado. Todo uso del lenguaje es **situado**, en el sentido de que el significado de una declaración puede depender de la situación en la cual se produce.
- La teoría del lenguaje formal y las gramáticas con **estructuras de frase** (y en particular, gramáticas **independientes del contexto**) son herramientas útiles para manejar algunos aspectos del lenguaje natural.
- Las oraciones en un lenguaje independiente del contexto pueden ser analizadas gramaticalmente en un tiempo $O(n^3)$ mediante un **analizador gramatical gráfico**.
- Es conveniente **aumentar** una gramática para manejar problemas tales como el sujeto (concordancia de verbo y caso pronominal). La **gramática causal definida** (DFG) es un formalismo que permite aumentaciones. Con DFG, la interpretación semántica y de análisis gramatical (e incluso la generación) se puede realizar usando inferencia lógica.
- La **interpretación semántica** puede ser también manejada con una gramática aumentada. Una forma casi-lógica puede ser intermediario útil entre árboles sintácticos y la semántica.
- La **ambigüedad** es un problema muy importante en la comprensión del lenguaje natural; la mayoría de las oraciones tienen varias posibles interpretaciones, pero

usualmente sólo una es apropiada. La desambiguación subyace en el conocimiento del mundo, de la situación actual, y del uso del lenguaje.

- La mayoría del lenguaje existe en un contexto de múltiples oraciones, no en una única. El **discurso** es el estudio de los textos conectados. Vimos cómo resolver las referencias pronominales a través de oraciones y cómo las oraciones se unen en segmentos coherentes.
- **Inducción gramatical**. La inducción gramatical puede aprender una gramática a partir de ejemplos, aunque hay limitaciones en cómo de bien la gramática generalizará.

Notas bibliográficas e históricas

SEMIÓTICA

El estudio de signos y símbolos como elementos del lenguaje fue denominado **semiótica** por John Locke (1690), aunque no fue desarrollado hasta el siglo XX (Peirce, 1902; de Saussure, 1993). Textos revisados recientemente incluyen Eco′s (1979) y Cobley's (1997).

La idea del lenguaje como acción proviene 20th-century lingüístico orientada la filosofía (Wittgenstein, 1953; Grice, 1957; Austin, 1962) y particularmente del libro *Speech Acts* (Searle, 1969). Un precursor de la idea actos de discurso fue Protagora's (s. 430 a.C.) que identificó cuatro tipos de sentencias: petición, pregunta, respuesta y prescripción. Cohen y Perrault (1979) sugirieron un modelo basado en plano de *speech acts*. Wilensky (1983), estudió conectar el lenguaje a la acción utilizando el reconocimiento del plan para comprender las historias. Cohen, Morgan y Pollack (1990) recogen un trabajo más reciente en esta área.

Al igual que las redes semánticas, las gramáticas de contexto libre (también conocidas como gramáticas de estructura de frases) son una reinvención de una técnica utilizada inicialmente por los ancianos gramatistas indios (especialmente Panini, s. 350 a.C.) studying Shastric Sanskrit (Ingerman, 1967). Fueron reinventadas por Noam Chomsky (1956) para el análisis de la sintaxis del inglés e independientemente por John Backus para el análisis de la sintaxis del Algol-58. La notación de Backus (Backus, 1996) ampliada por Naur (1963) es ahora calificada con la «N» en BNF, que originalmente se conocía por «Forma Normal Backus». Knuth (1968) definió una clase de gramática llamada gramática aumentada del atributo que es útil para los lenguajes de programación. Las gramáticas clausales definidas fueron introducidas por Colmerauer (1975) y desarrolladas y popularizadas por Pereira y Warren (1980). El lenguaje de programación Prolog fue inventado por Alain Colmerauer específicamente para el problema del emparejamiento del lenguaje francés. Colmerauer introdujo realmente un formalismo llamado gramática de la metamorfosis que fue más allá de las cláusulas definidas, pero las DCG le siguieron al poco tiempo.

Ha habido muchas tentativas de escribir gramáticas formales de lenguajes naturales, en lingüística «pura» y en lingüística de cómputo. Las gramáticas orientadas hacia la máquina incluyen aquellos desarrollados en el Linguistic String Project en la Universidad de Nueva York (Sager, 1981) y el proyecto XTAG en la Universidad de Pennsylvania (Doran *et al*, 1994). Un buen ejemplo de un sistema DCG moderno es el Core Language Engine. Hay varias gramáticas comprensivas pero informales del inglés (Jespersen,

1965; Quirk *et al*., 1985; McCawley, 1988; Huddleston y Pullum, 2002). Los buenos libros de textos en lingüística incluyen la introducción de Sag y Wasow (1999) a los textos de sintaxis y semántica de Chierchia y de McConnell-Ginet (1990) y los de Heim y Kratzer (1998). El texto de McCawley (1993) se concentra en la lógica para los lingüistas.

Desde mediados de los años 80, ha habido una tendencia a poner más información en el léxico y menos en la gramática. La gramática léxico-funcional, o LFG, (Bresnan, 1982) es la primera gramática formal importante que fue lexicalizada. Si se lleva la lexicalización hasta el extremo, se acaba en una **gramática de categorías**, que tiene solamente dos reglas gramaticales, o en una **gramática de dependencia** (Melcuk y Polguer, 1988), en la que no hay frases, solamente palabras. Sleator y Temperley (1993), describen un popular analizador gramatical que utiliza dependencias gramaticales. La gramática *Tree-adjoining* (*de árboles contiguos*), o TAG, (Joshi, 1985) no es estrictamente léxica, pero su forma de lexicalización está ganando en popularidad (Schabes *et al*., 1988). *Wordnet* (Fellbaum, 2001) es un diccionario disponible públicamente con cerca de 100.000 palabras y frases, clasificadas por categorías de partes de oraciones y con enlaces semánticos tales como sinónimos, antónimos y partes-de.

Los primeros algoritmos de análisis sintáctico para computador, los demostró Yngve (1955). A finales de 1960 se desarrollaron algoritmos eficientes, que han sufrido pocos cambios desde entonces (Kasami, 1965; Younger, 1967; Graham *et al*., 1980). Nuestro algoritmo de grafos fue concluido por Earley's (1970). En el texto de análisis sintáctico y compilación de Aho y Ullman (1972) aparece un buen resumen del algoritmo. Maxwell y Kaplan (1993) muestran cómo un analizador sintáctico de grafos con algunas ampliaciones puede ser eficiente en la mitad de los casos. Church y Patil (1982) dirigieron la resolución de la ambigüedad sintáctica.

La interpretación semántica formal de los lenguajes naturales tiene su origen en el campo de la filosofía y de la lógica formal y guarda una estrecha relación con la obra de Alfred Tarski (1935) sobre la semántica de los lenguajes formales. Bar-Hillel fue el primero en considerar los problemas de la pragmática y en proponer la forma de manejarla a través de la lógica formal. Por ejemplo, introdujo el término *indexical* (*indexación*) de C.S. Peirce (1902) en la lingüística (Bar-Hillel, 1954). El ensayo de Richard Montague «*English as a formal language*» (1970) es una especie de manifiesto del análisis lógico del lenguaje, pero el libro de Dowty *et al*. (1991) y el artículo de Lewis (1972) son más accesibles. Thomason (1974) editó las obras completas de Montague. En el campo de la inteligencia artificial, el trabajo de McAllester y Givan (1992) continúa con la tradición de Montagovian, añadiendo nuevas aclaraciones técnicas.

La idea de usar una lógica intermedia o cuasi-lógica para resolver problemas tales como la determinación del alcance de los cuantificadores se remonta a Woods (1978) y está presente en muchos sistemas recientes (Alshawi, 1992; Hwang y Schubert, 1993).

El primer sistema PLN (*Procesamiento de Lenguaje Natural*) para resolver una tarea actual fue probablemente el sistema de preguntas y respuestas BASEBALL (Green *et al*., 1961), que maneja preguntas sobre una base de datos estadística de béisbol. Justo después apareció el sistema LUNAR de Wood (1973), que responde preguntas sobre las rocas que se trajeron en el programa *Apollo*. Roger Schank y sus estudiantes construyeron una serie de programas (Schank y Abelson, 1977; Wilensky, 1978; Schank y Riesbeck, 1981; Dyer, 1983) que tratan sobre la tarea de comprender un lenguaje. Sin em-

bargo, se puso mayor énfasis en la representación y el razonamiento que en el lenguaje por sí mismo. Los problemas incluían la representación de situaciones estereotipadas (Cullingford, 1981) describiendo la organización de la memoria humana (Rieger, 1976; Kolodner, 1983) y la comprensión de planes y objetivos (Wilensky, 1983).

Desde los comienzos de las máquinas traductoras en los años 50, se consideró la generación de lenguaje natural, pero no se consideró como un asunto monolingüe hasta los 70. Son representativos los trabajos de Simmons y Slocum (1972) y Goldman (1975). PENMAN (Bateman y otros, 1989) fue uno de los primeros sistemas de generación completo basado en la Gramática Sistemática (Kasper, 1988). Desde los 90 están disponibles dos importantes sistemas de generación de dominio público, KPML (Bateman, 1997) y FUF (Elhadad, 1993). Entre los libros importantes sobre generación se incluye McKeown (1985), Hovy (1988), Pattern (1988) y Reiter y Dale (2000).

Uno de los primeros trabajos sobre desambiguación fue la teoría de *semántica de prioridades* de Wilk (1975), que trataba de encontrar interpretaciones que minimicen el número de anomalías semánticas. Hirst (1987) describe un sistema con objetivos similares que se aproxima bastante a la semántica de composición descrita en este capítulo. Hobbs *et al*. (1993) describen una red cuantitativa para medir la calidad de una interpretación sintáctica y semántica. Desde entonces, ha llegado a ser más común utilizar una red bayesiana explícita (Charniak y Goldman, 1992; Wu, 1993). En lingüística, la teoría de optimalidad[19] o perfeccionamiento (Linguistics) (Kager, 1999) se basa en la idea de la construcción de restricciones suaves sobre la gramática, obteniendo así un *ranking* natural de interpretaciones, que en la gramática genere todas las posibilidades de igual rango. Norving (1988) trata los problemas de considerar múltiples interpretaciones simultáneamente, mejor que establecer una única interpretación de máxima probabilidad. Los críticos literarios (Empson, 1953; Hobbs, 1990) han sido ambiguos con respecto a si la ambigüedad se debe resolver o cuidar.

Nunberg (1979) perfiló un modelo formal de metonimia. Lakoff y Johnson (1980) realizan un análisis y una catalogación atractivos de las metáforas comunes en inglés. Ortony (1979) presenta una colección de artículos sobre metáforas; Martin (1990) ofrece una aproximación computacional a la interpretación de metáforas.

El tratamiento de la resolución de referencias sigue a Hobbs (1978) en sus planteamientos. Lapppin y Leass (1994) construyen un modelo más complejo basado en un mecanismo de puntuación cuantitativo. Trabajos más recientes (Kehler, 1997; Ge *et al*., 1998) han utilizado una máquina de aprendizaje para sincronizar los parámetros cuantitativos. Los libros de Hirst (1981) y Mitkov (2002) son dos excelentes recopilaciones de resolución de referencias.

En 1758, David Hume en *Enquiry Concerning the Human Understanding* argumentó que el lenguaje está conectado por «tres principios de conexión entre ideas, denominadas *Semejanza, Contigüidad* en tiempo o lugar, y *Causa o Efecto*». Así empezó una larga historia de pruebas para definir relaciones de coherencia. Hobbs (1990) nos proporciona la colección usada en este capítulo; Mann y Thompson (1983) proporciona una colección más elaborada que incluye solución oculta, evidencia, justificación, motivación, razón, secuencia, posibilidad, elaboración, replanteamiento, condición, circuns-

[19] Se usa optimalidad, a pesar de no ser un término español, para referirse a una técnica de perfeccionamiento.

tancia, causa, concesión, ambientación, y tesis-antítesis. Este modelo evoluciona sobre una teoría de la estructura retórica (RST), que es, quizá hasta hoy la teoría más destacada (Mann y Thompson, 1988). Este capítulo toma prestado algunos de los ejemplos del capítulo de Jurafsky y Martin (2000) escrito por Andrew Kehler.

Grosz y Sidner (1986) presentan una teoría de discurso coherente basado en el desplazamiento del foco de atención, y Grosz *et al.* (1995) presentan una teoría relacionada que se basa en la noción del centrado. Joshi, Webber y Sag (1981) recopilan trabajos importantes sobre el discurso. Webber presenta un modelo de la interacción de las restricciones de sintaxis y del discurso, sobre lo que se puede decir en un momento del mismo (1983) y sobre la forma en la que los tiempos verbales interactúan con el discurso (1988).

El primer resultado importante sobre **gramática de inducción** fue negativo: Gold (1967) mostró que no es posible asegurar el aprendizaje de una gramática independiente del contexto correcta, proporcionando una colección de cadenas de la gramática. Esencialmente, la idea es que, dada una colección de cadenas $s_1, s_2, \ldots s_n$, la gramática correcta podría ser totalmente inclusiva ($S \rightarrow palabra^*$), o podía ser una copia de la entrada ($S \rightarrow s_1 \mid s_2 \mid \ldots \mid s_n$), o las dos cosas. Lingüistas eminentes, como Chomsky (1957, 1980) y Pinker (1989,2000), usaron los resultados de Gold para argumentar que debe existir una **gramática universal** innata que todos los niños conocen desde que nacen. El argumento conocido por la *Pobreza de los Estímulos*, es que los niños no reciben otras entradas de lenguaje que ejemplos positivos: sus padres producen ejemplos precisos de su lenguaje y muy raras veces corrigen errores. Por tanto, y dado que Gold demostró que el aprendizaje de una GIC a partir de ejemplos positivos es imposible, los niños ya deberían conocer la gramática y simplemente se debería ajustar alguno de los parámetros de su gramática innata y enseñarles vocabulario. Mientras que estos argumentos dominan las teorías de muchos de los lingüistas seguidores de Chomsky, estos han sido desestimados por otros lingüistas (Pullum, 1996; Elman *et al.*, 1997) y por la mayoría de los científicos de la computación, tan pronto como en 1969, Horning mostró que es posible aprender, en el sentido del aprendizaje PAC[20], una gramática independiente del contexto *probabilista*. Desde entonces, ha habido muchas demostraciones empíricas convincentes sobre el aprendizaje únicamente a través de ejemplos positivos, tales como el trabajo ILP de Mooney (1999), Muggleton (1996), Raedt (1994) y la remarcable tesis de Schütze (1995) y de Marcken (1996). Es posible aprender otros formalismos de gramáticas, como los lenguajes regulares (Oncina y García, 1992; Denis, 2001), y lenguajes de árboles regulares (Carrasco *et al.*, 1998) y autómata de estado finito (Parekh y Honavar, 2001).

El sistema SEQUITUR se debe a Nevill-Manning y a Witten (1997). Es interesante notar que, al igual que Marcken, ellos remarcan que sus esquemas de gramática de inducción son también buenos esquemas de comprensión. Esto está de acuerdo con el principio de la codificación de longitud de descripción mínima: una buena gramática es una gramática que minimiza la suma de dos longitudes: la longitud de la gramática y la longitud del árbol de análisis sintáctico del texto.

La programación lógica inductiva trabaja para que el aprendizaje del lenguaje incluya el sistema CHILL de Zelle y Mooney (1996) y un programa de Mooney y Califf (1995), que aprende reglas a partir del tiempo pasado de los verbos mejor que la red de nervios

[20] PAC – Probablemente Aproximadamente Correcto.

pasado o sistemas de árboles de decisión. Cussens y Dzeroski (2000) editaron una colección de ponencias sobre el lenguaje de aprendizaje en lógica.

La Asociación de Lingüística Computacional (ACL) realiza regularmente conferencias y publica la revista *Lingüística Computacional*. También hay una Conferencia Internacional sobre Lingüística Computacional (COLING).

Readings in Natural Languaje Processing (Grosz *et al*., 1986) es una antología que contiene muchas de las primeras ponencias importantes. Dale *et al*. (2000) hace hincapié en herramientas prácticas para construir sistemas PLN. El libro de texto de Jurafsky y Martin (2000) presenta una introducción comprensiva sobre este campo. Allen (1995) es una interpretación ligeramente antigua. Pereira y Sheiber (1987) y Convington (1994) ofrece revisiones concisas del procesamiento sintáctico basado en implementaciones de Prolog. La *Enciclopedia de IA* tiene muchos artículos útiles en este campo; *véanse* especialmente las entradas «Lingüística Computacional» y «Comprensión del Lenguaje Natural».

EJERCICIOS

22.1 Lea una sola vez el siguiente texto, tratando de comprender y recordar lo más que pueda. Posteriormente se le hará una prueba sobre él.

> El procedimiento es en realidad muy sencillo. Primero, distribuya todas las cosas en grupos distintos. Desde luego que una pila será suficiente dependiendo de todo lo que haya que hacer. Si no cuenta con los elementos necesarios, el siguiente paso consistiría en ir a otro sitio donde sí los hubiera; de no ser este el caso, todo estaría listo para que empiece. Es importante que no se exceda en realizar cosas. Es decir, es mejor hacer unas cuantas cosas a la vez, y no demasiadas. A corto plazo esto no parecería ser muy importante, pero podría ser causa de complicaciones. Cometer errores también resulta costoso. Al principio, el procedimiento completo podría parecer complicado. Sin embargo, al poco tiempo se convertirá en parte de la vida. Aunque es difícil prever que ya no se necesite la realización de esta labor en el futuro inmediato, uno nunca sabe. Una vez concluido el procedimiento, se vuelve a poner el material en distintos grupos. Se coloca en el lugar apropiado. Finalmente este material se utilizará una vez más y el ciclo se repite de nuevo. Sin embargo, es parte de la vida.

22.2 Utilizando la notación DCG escriba una gramática para un lenguaje que sea justamente como ε_1, exceptuando que fuerza la concordancia entre el sujeto y el verbo de una sentencia y, por tanto, no genera «Yo olfatean el *wumpus*».

22.3 Aumente la gramática ε_1, para que controle la concordancia entre el artículo y el nombre. Es decir, asegure qué «agentes» hay en una *FN*, y cuáles no.

22.4 Resuma las principales diferencias entre Java (o cualquier otro lenguaje de computación con el que esté familiarizado) y el inglés, comentando el problema de la «comprensión» en cada caso. Considere aspectos tales como la gramática, la sintaxis, la semántica, la pragmática, la composición, la dependencia del contexto, la ambigüedad léxica, la ambigüedad sintáctica, la localización de referencias (incluidos los pronombres), el conocimiento previo, y, en primer lugar, qué se entiende por «comprender».

22.5 ¿Cuáles de las siguientes razones justifican el empleo de una forma cuasi lógica?

a) Facilitar la elaboración de reglas de gramática por composición sencillas.
b) Ampliar la expresividad del lenguaje de representación semántica.
c) Capacidad para presentar ambigüedades de alcance de los cuantificadores (entre otros) de manera sucinta.
d) Facilitar la desambiguación semántica.

22.6 Determine qué interpretación semántica da a las siguientes oraciones utilizando la gramática propuesta en este capítulo:

a) Es un *wumpus*.
b) El *wumpus* está muerto.
c) El *wumpus* está en 2,2.

¿Sería una buena idea que la interpretación semántica de «Es un *wumpus*» se redujera a $\exists x\ x \in Wumpus$? Considere otras oraciones posibles tales como «Era un *wumpus*».

22.7 Sin consultar el Ejercicio 22.1 responda a las siguientes preguntas:

a) ¿Cuáles son los cuatro pasos que se mencionan?
b) ¿Qué paso no se incluye?
c) ¿Cuál es el «material» que se menciona en el texto?
d) ¿Qué clase de error resultaría costoso?
e) ¿Es mejor cometer muchos o pocos? ¿Por qué?

22.8 Este ejercicio se refiere a gramáticas de lenguajes muy sencillos

a) Escriba una gramática de contexto libre para el lenguaje $a^n b^n$.
b) Escriba una gramática de contexto libre para el lenguaje de los palíndromos: el conjunto de todas las cadenas cuya segunda mitad es el inverso de la primera.
c) Escriba una gramática de *context-sensitive* para el lenguaje de la duplicación: el conjunto de todas las cadenas cuya segunda mitad es la misma que la primera.

22.9 Considere la oración «Una persona se dirigió caminando lentamente hacia el supermercado» y el siguiente lexicon

Pronombre → **alguien**	*V* → **camino**
Adv → **lentamente**	*prep.* → **a**
Det → **el**	*Nombre* → **supermercado**

¿Cuál de los tres siguientes conjuntos de reglas gramaticales, genera la oración dada? Muestre el árbol de análisis sintáctico correspondiente.

(*A*):	(*B*):	(*C*):
S → *FN FV*	*S* → *FN FV*	*S* → *FN FV*
FN → *Pronombre*	*FN* → *Pronombre*	*FN* → *Pronombre*
FN → *Artículo Nombre*	*FN* → *Nombre*	*FN* → *Artículo FN*
FV → *FV FP*	*FN* → *Artículo FN*	*FV* → *Verbo Adv*
FV → *FV Adv Adv*	*FV* → *Verbo Vmod*	*Adv* → *Adv Adv*
FV → *Verbo*	*Vmod* → *Adv Vmod*	*Adv* → *FP*
FP → *Prep. FN*	*Vmod* → *Adv*	*FP* → *Prep. FN*
FN → *Nombre*	*Adv* → *FP*	*FN* → *Nombre*
	FP → *prep. FN*	

Para cada una de las tres gramáticas anteriores, escriba tres oraciones en español y tres oraciones no españolas que genere la gramática. Cada una de las oraciones debería ser significativamente diferente, debería tener al menos seis palabras y se debería basar en un conjunto nuevo completo de entradas léxicas (las cuales debería definir). Sugiera de qué forma se debe ampliar cada gramática para permitir la generación de oraciones no españolas.

22.10 Implemente una versión del algoritmo de análisis con grafo que devuelva un árbol empaquetado con todos los arcos que abarquen la entrada completa.

22.11 Implemente una versión del algoritmo de análisis con grafo que devuelva un árbol empaquetado con el arco más a la izquierda el más largo, y si ese arco no puede abarcar la entrada completa, continúe el análisis desde el final de ese arco. Muestre por qué necesitará llamar a PREDECIR antes de continuar. El resultado final es una lista de árboles empaquetados para que la lista completa abarque la entrada.

22.12 (Derivado de Barton *et al*. (1987).) Este ejercicio se refiere a un lenguaje que se llama *Bufalo*^{n}, que se parece bastante al inglés (o al menos a ε_0), excepto porque la única palabra de este léxico es *bufalo*. Las siguientes son dos sentencias del lenguaje:

Bufalo bufalo bufalo Bufalo bufalo
Bufalo Bufalo bufalo bufalo bufalo Bufalo bufalo

En el caso de que crea que no son sentencias, las siguientes son dos sentencias en inglés con una estructura sintáctica correspondiente a las anteriores:

Dallas cattle bewilder Denver cattle
Chefs London critics admire cook French food

Escriba la gramática de *Bufalo*^{n}. Las categorías léxicas son ciudad, nombre plural y verbo transitivo, y habría una regla gramatical para la sentencia, una para la frase verbal y tres para la frase nominal: nombre plural, frase nominal precedida por una ciudad como un modificador y una frase nominal seguida por una cláusula relativa reducida. Una cláusula relativa reducida es una cláusula que pierde el pronombre relativo. Además, la clausula está formada por una frase nominal sujeto seguida por un verbo sin un objeto. Un ejemplo de cláusula relativa reducida, en el ejemplo anterior, es «London critics admire». Tabule el número de analizadores posibles para *Bufalo*^{n} desde *n* hasta 10.
Extra: Carl de Marcken calculó que hay 121,030,872,213,055,159,681,184,485 sentencias de longitud 200 para *Bufalo*^{n} (para la gramática que él usó). ¿Cómo lo hizo?

22.13 Dibuje un árbol de análisis sintáctico para un discurso correspondiente a la historia de la página 933 sobre cómo John va a un restaurante de lujo. Use las dos reglas gramaticales para *Segmento*, estableciendo la propiedad *RelaciónCoherencia* para cada nodo. (Necesita mostrar el análisis para las sentencias individuales.) Haga lo mismo para las sentencias de la 5 a la 10 del discurso elegido.

22.14 Se nos olvidó mencionar que el texto del Ejercicio 22.1 se titula «Lavando trapos». Relea el texto y responda a las preguntas del Ejercicio 22.7. ¿Lo hace mejor esta vez? Bransford y Johnson (1973) usaron este texto en un experimento mejor controlado y consideraron que ese título ayudaba de forma significativa. ¿Qué le dice sobre la comprensión del discurso?

23 Procesamiento probabilístico del lenguaje

En el cual vemos cómo se pueden utilizar modelos del lenguaje simples, entrenados estadísticamente para procesar colecciones de millones de palabras, en vez de sencillas oraciones exactas.

En el Capítulo 22, vimos cómo un agente podía comunicarse con otro agente (ser humano o *software*), a través de expresiones en un lenguaje común. Para extraer el significado total de las expresiones es necesario realizar el análisis sintáctico y semántico completo de dichas expresiones, y esto es posible porque las palabras son cortas y restringidas a un dominio limitado.

BASADO EN SU CORPUS

En este capítulo, consideramos el acercamiento a la comprensión de idiomas **basado en su corpus**. Un corpus (*corpora* en plural) es una colección grande de texto, tal como los 1.000 millones de páginas que constituyen el World Wide Web. El texto está escrito por y para los seres humanos, y la tarea del *software* es hacerlo más fácil para que el ser humano encuentre la información correcta. Esta aproximación implica el uso de la estadística y aprender de su corpus, y exige generalmente modelos probabilísticos del lenguaje que pueden aprenderse de los datos y que son más simples que el DCGs aumentado del Capítulo 22. Para la mayoría de las tareas, el volumen de datos más que otras cosas afecta al modelo más simple del lenguaje. Veremos tres tareas específicas: recuperación de datos (Apartado 23.2), extracción de la información (Apartado 23.3) y traducción automática (Apartado 23.4). Pero primero presentamos una visión general de los modelos probabilísticos del lenguaje.

23.1 Modelos probabilísticos del lenguaje

El Capítulo 22 nos proporcionó un modelo lógico del lenguaje: utilizamos CFGs y DCGs para caracterizar una cadena como un miembro o no de un lenguaje. En esta apar-

tado introduciremos varios modelos probabilísticos. Los modelos probabilísticos tienen varias ventajas. Pueden ser entrenados convenientemente a partir de los datos: el aprendizaje es justo una cuestión de contar ocurrencias (con algunas relajaciones por los errores debidos a basarnos en muestras de tamaño pequeño). También son más robustos (porque pueden aceptar cualquier cadena, a pesar de tener una probabilidad baja), reflejan el hecho de que no todos los hablantes están de acuerdo en qué frases son parte de un idioma actualmente, y pueden utilizarse para la desambiguación: la probabilidad de elegir la interpretación más probable.

MODELO PROBABILÍSTICO DEL LENGUAJE

Un modelo probabilístico del lenguaje define una distribución de la probabilidad sobre un conjunto (posiblemente infinito) de cadenas. Ya hemos visto ejemplos de modelos: el bigram y el trigram, son modelos del lenguaje usados en el reconocimiento del habla (Apartado 15.6). Un modelo unigram asigna una probabilidad $P(w)$ a cada palabra del léxico. El modelo asume que las palabras están elegidas independientemente, así que la probabilidad de una secuencia es el producto de la probabilidad de sus palabras, dada por $\Pi_i\, P(w_i)$. La secuencia siguiente de 20 palabras fue generada al azar a partir de un modelo unigram de las palabras de este libro:

> lógico sea al igual que la confusión a puede enderezar el agente de los intentos que es la meta era

Un modelo bigram asigna una probabilidad $P(w_i \mid w_{i-1})$ a cada palabra, dependiendo de la palabra anterior. La Figura 15.21 enumeró algunas de estas probabilidades del bigram. Un modelo bigram de este libro genera al azar la siguiente secuencia:

> planeando los sistemas expertos puramente de diagnóstico son acercamiento de cómputo muy similar serían representados compacto usando el dedo

En general, un modelo n-gram condiciona una palabra a las $n - 1$ anteriores, asignando una probabilidad $P(w_i \mid w_{i-(n-1)} \ldots w_{i-1})$. Un modelo trigram de este libro genera al azar la siguiente secuencia:

> planificación y programación se integran el éxito de bayes simple que el modelo es justo una fuente anterior posible para

Incluso con esta pequeña muestra, debería estar claro que el modelo trigram es mejor que el modelo bigram (que es mejor que el modelo unigram), para aproximar la lengua inglesa y para aproximar el contenido de un libro de textos de IA. Los modelos asimismo convienen: el modelo trigram asigna a su secuencia al azar una probabilidad de 10^{-10}, el bigram 10^{-29}, y el unigram 10^{-59}.

Con medio millón de palabras, este libro no contiene bastantes datos para producir un buen modelo bigram, permitiendo solamente un modelo trigram. En el léxico de este libro hay cerca de 15.000 palabras distintas, así que el modelo bigram incluye $15.000^2 = 225$ millones de pares de palabras. Claramente, por lo menos un 99,8 por ciento de estos pares tendrán una frecuencia de aparición cero, pero no queremos que nuestro modelo diga que todos estos pares son imposibles. Necesitamos alguna manera de **suavizar** las frecuencias con valor cero. La manera más simple de hacer esto se llama **suavizar añadiendo uno**: agregamos uno al contador de cada bigram posible. Si en la recopilación hay N palabras y B bigrams posibles, entonces a cada bigram con un contador real c se le asigna una estimación de probabilidad de $(c + 1)/(N + B)$. Este méto-

SUAVIZAR

SUAVIZAR AÑADIENDO UNO

do elimina el problema de los n-gram de probabilidad cero, pero la asunción de que cada contador sea incrementado exactamente en uno es dudosa y puede conducir a estimaciones pobres.

SUAVIZADO POR INTERPOLACIÓN LINEAR

Otra aproximación es el **suavizado por interpolación lineal**, que combina los modelos trigram, bigram, y unigram por interpolación lineal. Definimos nuestra estimación de la probabilidad como

$$\hat{P}(w_i \mid w_{i-2}w_{i-1}) = c_3 P(w_i \mid w_{i-2}w_{i-1}) + c_2 P(w_i \mid w_{i-1}) + c_1 P(w_i),$$

donde $c_3 + c_2 + c_1 = 1$. El c_i de los parámetros puede ser fijo, o pueden ser entrenados con un algoritmo EM. Es posible tener valores de los c_i que dependan de los contadores n-gram, de modo que pongamos un peso más alto en las estimaciones de la probabilidad que se derivan de registros más altos.

Un método para evaluar un modelo del lenguaje es como sigue: primero, dividir su corpus en un corpus de entrenamiento y un corpus de prueba. Determinar los parámetros del modelo con base en los datos del entrenamiento. Entonces calcular la probabilidad asignada al corpus de prueba por el modelo; cuanto más alta la probabilidad mejor. Un problema con esta aproximación es que $P(words)$ es demasiado pequeña para las cadenas largas; los números podrían causar desbordamiento de capacidad inferior de punto flotante, o podrían ser difíciles de leer.

PERPLEJIDAD

En vez de probabilidad podemos calcular la **perplejidad** de un modelo en una cadena de prueba de palabras

$$Perplejidad(words) = 2^{-\log_2(P(words))/N}$$

donde N es el número de palabras. Cuanto más baja es la perplejidad, mejor es el modelo. Un modelo n-gram que asigna a cada palabra una probabilidad de $1/k$ tendrá perplejidad k; se puede pensar en perplejidad como el factor de ramificación medio.

SEGMENTACIÓN

Como ejemplo de qué puede hacer el modelo n-gram, consideremos la tarea de **segmentación**: encontrar los límites de palabras en un texto sin espacios. Esta tarea es necesaria en japonés y chino, idiomas que se escriben sin espacios entre las palabras, pero asumimos que la mayoría de los lectores se sentirán más cómodos con inglés. La oración

Itiseasytoreadwordswithoutspaces

para nosotros es de hecho fácil de leer. Puede ser que se piense que es porque tenemos conocimiento completo de la sintaxis, semántica y de la práctica de inglés. Mostraremos que la oración se puede descifrar fácilmente por un modelo simple de palabra unigram.

Vimos anteriormente cómo la ecuación de Viterbi (15.9), se puede utilizar para solucionar el problema de encontrar la secuencia más probable a través de un retículo de las posibilidades de la palabra. La Figura 23.1 muestra una versión del algoritmo de Viterbi diseñado específicamente para el problema de la segmentación. Toma como entrada una distribución de probabilidad de la palabra del unigram, $P(word)$, y una cadena. Entonces, para cada posición i de la cadena, almacena en $best[i]$ la probabilidad de la se-

```
función SEGMENTACION-VITERBI(text, P) devuelve las mejores palabras y sus probabilidades
  entradas: text, una cadena de caracteres con los espacios eliminados
            P distribución de la probabilidad para las palabras del unigram

  n ← LENGTH(text)
  words ← vector vacío de longitud n + 1
  best ← vector de longitud n + 1, inicialmente todas las componentes a 0.0
  best[0] ← 1.0
  / * complete los vectores best y words vía programación dinámica * /
  para i = 0 hasta n hacer
    para j = 0 hasta i − 1 hacer
      word ← text[ j:i]
      w ← LENGTH(word)
      si P[word] × best[i − w] ≥ best[i] entonces
        best[i] ← P[word] × best[i − w]
        words[i] ← word
  /* ahora recupere la secuencia de las mejores palabras */
  sequence ← lista vacía
  i ← n
  mientras i > 0 hacer
    poner words[i] en el frente de sequence
    i ← i − LENGTH(words[i])
  / * Retorna la secuencia de las mejores palabras y la probabilidad de la secuencia * /
  devolver sequence, best[i]
```

Figura 23.1 Algoritmo basado en el de Viterbi para segmentación palabras. Dada una cadena de palabras con los espacios eliminados, recupera la segmentación más probable en palabras.

cuencia más probable que la atraviesa desde el comienzo hasta el i. También almacena en $words[i]$ la conclusión de la palabra en la posición i que produjo la mejor probabilidad. Una vez que haya acumulado los vectores *best* y *words* mediante programación dinámica, entonces trabaja al revés con palabras para encontrar el mejor camino. En este caso, con el modelo del unigram del libro, la mejor secuencia de palabras «Is easy to read words without spaces» con probabilidad 10^{-25}. Comparando las partes de la secuencia, vemos por ejemplo que «easy» tiene probabilidad $2{,}6 \times 10^{-4}$ en el unigram, mientras que «e as y» tiene una probabilidad mucho más baja, $9{,}8 \times 10^{-12}$, a pesar del hecho de que las palabras *e* e *y* son bastante comunes en las ecuaciones del libro. De forma similar, tenemos

$$P(\text{«without»}) = 0{,}0004;$$
$$P(\text{«with»}) = 0{,}005;\ P(\text{«out»}) = 0{,}0008;$$
$$P(\text{«with out»}) = 0{,}005 \times 0{,}0008 = 0{,}000004.$$

Por tanto «without» es 100 veces más probable que «with out» según el modelo unigram.

En esta apartado hemos discutido modelos n-gram sobre palabras, pero hay también muchas aplicaciones de los modelos n-gram sobre otras unidades, tales como caracteres o partes del habla.

Gramáticas probabilísticas independientes del contexto

GRAMÁTICAS PROBABILÍSTICAS INDEPENDIENTES DEL CONTEXTO

Los modelos probabilísticos *n*-gram de las gramáticas independientes del contexto se aprovechan de la estadística de co-ocurrencia en los corpus, pero no aportan ninguna noción de la gramática para valores mayores que *n*. Un modelo alternativo del lenguaje es la **gramática probabilística independiente del contexto**, o PCFG[1], que consiste en un CFG en donde cada regla de reescritura tiene asociada una probabilidad. La suma de las probabilidades a través de las reglas con el mismo lado izquierdo es 1. La Figura 23.2 muestra un PCFG para una parte de la gramática ε_0.

En el modelo PCFG, la probabilidad de una secuencia, *P*(*words*), es justo la suma de las probabilidades de sus árboles de análisis. La probabilidad de un árbol dado es el producto de las probabilidades de todas las reglas que constituyen los nodos del árbol. La Figura 23.3 muestra cómo calcular la probabilidad de una oración. Es posible calcular esta probabilidad usando un programa de interpretación del grafo CFG para enumerar las posibles interpretaciones y después simplemente añadirlo a las probabilidades. Sin embargo, si estamos interesados solamente en el análisis más probable entonces es un derroche la enumeración de los improbables. Podemos utilizar una variación del algoritmo de Viterbi para encontrar eficientemente el análisis más probable, o podemos utilizar una técnica de búsqueda mejor el primero (tal como lo A*).

S → *FN FV* [1.00]

FN → *Pronombre* [0.10]
| *Nombre* [0.10]
| *Sustantivo* [0.20]
| *Articulo Sustantivo* [0.50]
| *FN PP* [0.10]

FV → *Verbo* [0.60]
| *FV FN* [0.20]
| *FV PP* [0.20]

PP → *Preposicion FN* [1.00]

Sustantivo → **brisa** [0.10] | ***wumpus*** [0.15] | **agente** [0.05] | …
Verbo → **ve** [0.15] | **huele** [0.10] | **van** [0.25] | …
Pronombre → **mi** [0.05] | **tú** [0.10] | **yo** [0.25] | **él** [0.20] | …
Articulo → **el** [0.30] | **un** [0.35] | **cada** [0.05] | …
Preposition → **a** [0.30] | **en** [0.25] | **sobre** [0.05] | …

Figura 23.2 Una gramática probabilística independiente del contexto (PCFG) y del léxico para una porción de la gramática ε_0. Los números entre corchetes indican la probabilidad de que un símbolo del lado izquierdo sea reescrito con la regla correspondiente.

[1] PCFG se conoce también como gramáticas estocásticas contexto libre SCFG.

Figura 23.3 Árbol de análisis para la oración «Cada *wumpus* huele» mostrando las probabilidades de cada subestructura. La probabilidad del árbol en su totalidad es $1{,}0 \times 0{,}5 \times 0{,}05 \times 0{,}15 \times 0{,}60 \times 0{,}10 = 0{,}000225$. Puesto que este árbol es el único análisis para la oración, ese número es también la probabilidad de la oración.

El problema con los PCFGs es que son contexto libre. Eso significa que la diferencia entre P(«eat a banana») y P(«eat a bandanna») depende solamente de P(«banana») frente a P(«bandanna») y no en la relación entre «eat» y los respectivos objetos. Para conseguir esa clase de relación, necesitaremos una cierta clase de modelo sensible al contexto, como el **PCFG léxico**, en el cual la cabeza de una frase[2] puede desempeñar un papel en la probabilidad de la frase que la contiene. Con bastantes datos de entrenamiento, podríamos hacer la regla $FV \leftarrow FV\ FN$ condicionada a la cabeza del FV (eat) y la cabeza del FN (banana). El PCFG léxico captura así algunas de las restricciones de la co-ocurrencia de los modelos del n-gram junto con las restricciones gramaticales de los modelos CFG.

Un problema más es que los PCFG tienen una preferencia demasiado fuerte por las oraciones más cortas. En un corpus tal como el diario *Wall Street Journal* la longitud media de una oración es aproximadamente de 25 palabras. Pero un PCFG terminará generalmente asignando una probabilidad bastante alta a reglas tales como $S \rightarrow FN\ FV$ y $FN \rightarrow Pronombre$ y $FV \rightarrow Verbo$. Esto significa que el PCFG asignará probabilidad bastante alta a muchas cadenas cortas, por ejemplo «él durmió», mientras que en el diario será más probable ver algo como «ha sido divulgada por una fuente fiable del gobierno que la alegación de que él durmió es creíble». Parece que las frases en el diario realmente no son de contexto libre; en vez de eso los escritores tienen una idea de la longitud prevista de la oración y utilizan esa longitud como restricción global en sus oraciones. Esto es difícil de reflejar en un PCFG.

Aprendizaje de probabilidades para PCFGs

Para crear un PCFG, tenemos toda la dificultad de construir un CFG, combinada con el problema de fijar las probabilidades para cada regla. Esto sugiere que aprender la gramática de los datos podría ser mejor que una aproximación con la ingeniería de conocimiento. Justo como con el reconocimiento del habla, hay dos tipos de datos que

[2] La cabeza de una frase es la palabra más importante, por ejemplo, el sustantivo de una oración nominal.

podrían darse: analizados y no analizados. La tarea es considerablemente más fácil si tenemos datos que han sido analizados en árboles por lingüistas (o por lo menos por nativos entrenados). Crear tal corpus es una inversión grande, y las más grandes contienen «solamente» alrededor de un millón de palabras. Dado un corpus de árboles, podemos crear un PCFG simplemente contando (y suavizando): para cada símbolo no terminal, considerando todos los nodos con ese símbolo como raíz, y creando las reglas para cada combinación diferente de hijos en esos nodos. Por ejemplo, si aparece el símbolo *FN* 100.000 veces, y hay 20.000 casos de *FV* en la lista de los hijos [*FN, PP*], entonces se crea la regla

$$FV \rightarrow FN\ PP\ [0.20]$$

La tarea es mucho más dura si todo lo que tenemos es texto no analizado. En primer lugar, tenemos realmente dos problemas: aprender la estructura de las reglas de la gramática y aprender las probabilidades asociadas a cada regla. (Tenemos la misma distinción para el aprendizaje en redes neuronales o en redes de Bayes.)

De momento asumiremos que la estructura de las reglas está dada y que sólo estamos intentando aprender las probabilidades. Podemos utilizar una aproximación de maximización de la esperanza (EM), tal como hicimos en el aprendizaje HMMs. Los parámetros que estamos intentando aprender son las probabilidades de la regla. Las variables ocultas son los árboles del análisis: no sabemos si una cadena de las palabras. $w_i \ldots w_j$ es o no es generada por una regla $X \rightarrow \alpha$. El paso E estima la probabilidad que cada sub-cadena sea generada por cada regla. Entonces el paso M estima la probabilidad de cada regla. El cómputo total se puede hacer mediante programación dinámica con un algoritmo llamado **algoritmo dentro-fuera** en analogía al algoritmo adelante-atrás para HMMs.

ALGORITMO DENTRO-FUERA

El algoritmo dentro-fuera parece mágico al inducir una gramática a partir de texto no analizado. Pero tiene varias desventajas. Primero, es lento: $O(n^3 t^3)$, donde n es el número de palabras en una frase y t es el número de símbolos no terminales. En segundo lugar, el espacio de las asignaciones de la probabilidad es muy grande, y empíricamente el problema de evitar máximos locales es un problema severo. Se pueden intentar alternativas tales como *simulated annealing*, con un coste incluso mayor. Tercero, los análisis que son asignados por las gramáticas resultantes son a menudo difíciles de entender y no satisfacen a los lingüistas. Esto hace complicado combinar conocimiento manual con la inducción automática.

Aprendizaje de la estructura de las reglas para PCFGs

Ahora suponga que no se sabe la estructura de las reglas de la gramática. Nuestro primer problema es que el espacio de los conjuntos posibles de la regla es infinito, así que no sabemos cuántas reglas considerar ni qué tamaño tiene cada regla. Una forma de simplificar este problema es aprender una gramática en la **forma normal de Chomsky**, que significa que cada regla está en una de las dos formas

FORMA NORMAL DE CHOMSKY

$$X \rightarrow Y\ Z$$
$$X \rightarrow \mathbf{t}\ ,$$

donde X, Y y Z son no terminales y **t** es un terminal. Cualquier gramática independiente del contexto se puede reescribir como una gramática en la forma normal de Chomsky que reconozca el mismo lenguaje. Podemos entonces restringirnos arbitrariamente a n símbolos no terminales, así tendríamos $n^3 + nv$ reglas, donde v es el número de símbolos terminales. En la práctica, este acercamiento se ha probado eficaz solamente para las gramáticas pequeñas. Una aproximación alternativa llamada **modelo Bayesiano combinado** es similar al modelo SEQUITUR (Apartado 22.8). La aproximación comienza construyendo los modelos locales (gramáticas) de cada oración y después utiliza una longitud mínima de la descripción para combinar modelos.

MODELO BAYESIANO COMBINADO

23.2 Recuperación de datos

RECUPERACIÓN DE DATOS

La **recuperación de datos** es la tarea de encontrar los documentos que son relevantes para los usuarios que necesitan información. Los ejemplos más conocidos de sistemas de recuperación de datos son los motores de búsqueda en el World Wide Web. Un usuario de WWW puede editar una consulta por ejemplo [libro de IA] en un motor de búsqueda y ver una lista de páginas relevantes. En esta apartado, veremos cómo se construyen tales sistemas. Un sistema de la recuperación de datos (en adelante RD) se puede caracterizar por:

1. **Una colección del documento.** Cada sistema debe decidir qué desea tratar como documento: un párrafo, una página, o un texto de múltiples páginas.
2. **Una pregunta hecha en un lenguaje de consulta.** La pregunta especifica lo que el usuario desea saber. El lenguaje de consulta puede ser justo una lista de palabras, por ejemplo [libro de IA]; o puede especificar una frase de las palabras que deben ser adyacentes, como en [«libro de IA»]; puede contener a operadores boleanos como en [IA *Y* libro]; puede incluir a operadores no-Booleanos por ejemplo [libro *NEAR* IA] o [libro IA *SITIO*:www.aaai.org].
3. **Un conjunto resultado.** Éste es el subconjunto de los documentos que el sistema RD juzga **relevantes** para la consulta. Por *relevante*, significamos probablemente ser utilizada por la persona que hizo la pregunta, por la necesidad de información particular expresada en la consulta.
4. **Una presentación del conjunto resultado.** Esto puede ser tan simple como una lista alineada de títulos de documento o tan complejo como un mapa de colores del conjunto resultado proyectado sobre un espacio tridimensional.

PREGUNTA

LENGUAJE DE CONSULTA

CONJUNTO RESULTADO

RELEVANTE

PRESENTACIÓN

Después de leer el capítulo anterior, uno podría suponer que un sistema de la recuperación de datos podría ser construido analizando la colección de documentos en una base de conocimiento de frases lógicas y después analizando cada consulta y preguntando a la base de conocimientos para las respuestas. Desafortunadamente, nadie ha tenido éxito total en la construcción de un sistema RD de gran escala. Es demasiado difícil construir un léxico y una gramática que cubra una gran colección del documento, por ello todos los sistemas RD utilizan modelos de lenguaje más simples.

MODELO BOOLEANO DE LA PALABRA CLAVE

Los primeros sistemas RD trabajaban con un **modelo booleano de la palabra clave**. Cada palabra en la colección del documento se trata como característica boleana que es verdad en un documento si la palabra se encuentra en el documento y falso si no lo hace. Por ello, la característica «recuperación» es verdad para el capítulo actual pero falsa para el Capítulo 15. El lenguaje de consulta es el lenguaje de las características formando expresiones booleanas. Un documento es relevante solamente si la expresión evalúa con verdad. Por ejemplo, la consulta [información *Y* recuperación] es verdad para el capítulo actual y falsa para el Capítulo 15.

Este modelo tiene la ventaja de ser simple de explicar e implementar. Sin embargo, tiene algunas desventajas. Primero, el grado de importancia de un documento es sólo una pequeña parte, así que no indica cómo ordenar los documentos relevantes para la presentación. En segundo lugar, las expresiones booleanas pudieran ser desconocidas por los usuarios que no sean programadores o matemáticos. Tercero, puede ser difícil formular una consulta apropiada, incluso para un usuario experto. Suponga que intentamos [información *Y* recuperación *Y* modelos *Y* optimización] y que obtenemos un conjunto vacío como resultado. Podríamos intentar [información *O* recuperación *O* modelos *U* optimización], pero si devuelve demasiados resultados, es difícil saber qué intentar después.

La mayoría de los modelos RD usan modelos basados en la estadística de contadores de palabra (y a veces otras características de bajo nivel). Explicaremos un marco probabilístico que encaje bien en los modelos de lenguaje que hemos cubierto. La idea clave es que, dada una consulta, deseamos encontrar los documentos que tienen más probabilidad de ser relevantes. Es decir deseamos calcular

$$P(R = verdad \mid D, Q)$$

donde D es un documento, Q es una consulta y R es una variable booleana aleatoria que indica importancia. Una vez que tengamos esto, podemos aplicar el principio de la graduación de la probabilidad, que dice que si tenemos presente el conjunto resultado como una lista ordenada, nosotros deberíamos hacerlo disminuyendo la probabilidad de importancia.

MODELADO DEL LENGUAJE

Hay varias maneras de descomponer la distribución conjunta $P(R = verdad \mid D, Q)$. Mostraremos la aproximación conocida como **modelado del lenguaje**, que estima un modelo del lenguaje para cada documento y entonces, para cada pregunta, computamos la probabilidad de la pregunta, dado el modelo del lenguaje de los documentos. Usando r para denotar el valor $R = verdad$, podemos reescribir la probabilidad como sigue:

$$\begin{aligned} P(r \mid D,Q) &= P(D, Q \mid r)\, P(r)/P(D, Q) \quad \text{(por regla de Bayes)} \\ &= P(Q \mid D, r)P(D \mid r)P(r)/P(D, Q) \quad \text{(por regla de la cadena)} \\ &= \alpha\, P(Q \mid D, r)P(r \mid D)/P(D, Q) \quad \text{(por regla de Bayes, para } D \text{ fijo)} \end{aligned}$$

Dijimos que intentábamos maximizar $P(r \mid D, Q)$, pero, equivalentemente, podemos maximizar el cociente de las probabilidades $P(r \mid D, Q)/P(\neg r \mid D, Q)$. Es decir, podemos ordenar los documentos basados en el resultado:

$$\frac{P(r \mid D, Q)}{P(\neg r \mid D, Q)} = \frac{P(Q \mid D, r)P(r \mid D)}{P(Q \mid D, \neg r)P(\neg r \mid D)}$$

Esto tiene la ventaja de eliminar el término $P(D, Q)$. Ahora asumiremos que para los documentos irrelevantes, el documento es independiente de la consulta. Es decir, si un documento es irrelevante para una pregunta, entonces sabiendo que el documento no ayudará a contestar la pregunta, se elimina de la consulta. Esta asunción se puede expresar por

$$P(D, Q \mid \neg r) = P(D \mid \neg r)\, P(Q \mid \neg r)$$

Con la asunción, conseguimos

$$\frac{P(r \mid D, Q)}{P(\neg r \mid D, Q)} = P(Q \mid D, r) \times \frac{P(r \mid D)}{P(\neg r \mid D)}$$

El factor $P(r \mid D)/P(\neg r \mid D)$ es la probabilidad independiente de la consulta de que el documento sea relevante. Ésta es una medida de la calidad del documento; algunos documentos son relevantes a cualquier consulta con más probabilidad, porque el documento es intrínsecamente de alta calidad. Para los artículos de revistas académicas podemos estimar la calidad por el número de citaciones, y para las páginas Web podemos utilizar el número de enlaces a la página. En cualquier caso, podríamos dar más peso a referencias de alta calidad. La antigüedad de un documento puede ser también un factor para estimar su importancia independientemente de la consulta.

El primer factor, $P(Q \mid D, r)$, es la probabilidad de que una consulta de un documento sea relevante. Para estimar esta probabilidad debemos elegir un modelo de lenguaje de cómo se relacionan las consultas con los documentos relevantes. Una opción corriente es representar los documentos con un modelo de palabras unigram. Esto también se conoce como modelo de RD **bolsa de palabras**, porque lo significativo es la frecuencia de cada palabra en el documento, no su orden. En este modelo los documentos (muy cortos) «hombre muerde perro» y «perro muerde hombre» se comportarán idénticamente. Claramente, *significan* cosas diferentes, pero es verdad que son relevantes a consultas acerca de perro y morder. Ahora para calcular la probabilidad de una consulta dado un documento relevante, multiplicamos las probabilidades de las palabras en la consulta, según el modelo unigram del documento. Éste es un modelo sencillo de Bayes. Usando Q_j para indicar la palabra j-ésima de la consulta, tenemos

BOLSA DE PALABRAS

$$P(Q \mid D, r) = \prod_j P(Q_j \mid D, r)$$

Esto nos permite que hacer la simplificación

$$\frac{P(r \mid D, Q)}{P(\neg r \mid D, Q)} = \Pi_j P(Q_j \mid D, r)\, \frac{P(r \mid D)}{P(\neg r \mid D)}$$

Por último, estamos preparados para aplicar estos modelos matemáticos a un ejemplo. La Figura 23.4 muestra la estadística del unigram para las palabras en la consulta [modelo de Bayes de recuperación de datos] sobre una colección del documento que consiste en cinco capítulos seleccionados de este libro. Asumimos que los capítulos son de calidad uniforme, así que estamos interesados solamente en calcular la probabilidad de la consulta dado el documento, para cada documento. Hacemos esto dos veces, una vez usando el estimador máximo de probabilidad D_i sin suavizar y otra vez usando la probabilidad un modelo D'_i con suavizado agregando uno. Se debe esperar que el capítulo

Palabras	Pregunta	Capítulo 1 Introducción	Capítulo 13 Incertidumbre	Capítulo 15 Tiempo	Capítulo 22 NLP	Capítulo 23 Actual
Bayes	1	5	32	38	0	7
Información	1	15	18	8	12	39
Recuperación	1	1	1	0	0	17
Modelo	1	9	7	160	9	63
N	4	14.680	10.941	18.186	16.397	12.574
$P(Q \mid D_i, r)$		$1{,}5 \times 10^{-14}$	$2{,}8 \times 10^{-13}$	0	0	$1{,}2 \times 10^{-11}$
$P(Q \mid D'_i, r)$		$4{,}1 \times 10^{-14}$	$7{,}0 \times 10^{-13}$	$5{,}2 \times 10^{-13}$	$1{,}7 \times 10^{-15}$	$1{,}5 \times 10^{-11}$

Figura 23.4 Un modelo probabilístico RD para la consulta [modelo Bayes recuperación datos] sobre una colección de documentos que consiste en cinco capítulos de este libro. Damos las frecuencias de apariciones de la palabra para cada par (documento, palabra) y N el contador total de palabras para cada documento. Utilizamos dos modelos de documentos (D_i es el modelo de palabras unigram no suavizado del documento i-ésimo, y D'_i es el mismo modelo con suavizado agregando uno) y calculamos la probabilidad de la consulta dado cada documento para ambos modelos. El capítulo actual (23) es el claro ganador, sobre 200 veces más probable que cualquier otro capítulo bajo cualquier modelo.

actual tenga una respuesta alta para esta consulta, y de hecho en cualquiera de los modelos ocurre.

El modelo suavizado tiene la ventaja de que es menos susceptible al ruido y que puede asignar una probabilidad distinta a cero de la importancia en un documento que no contenga todas las palabras. El modelo sin suavizar tiene la ventaja de que es más fácil calcular para las colecciones con muchos documentos: si creamos un índice que liste los documentos que mencionan cada palabra, después podemos generar rápidamente un resultado intersectando estas listas, y solamente necesitamos calcular $P(Q \mid D_i)$ para los documentos de la intersección, en vez de para cada documento.

Evaluación de los Sistemas de RD

¿Cómo sabemos si un sistema RD está bien realizado? Emprendemos un experimento en el cual el sistema realice un conjunto de consultas y los conjuntos resultado son puntuados con respecto a criterios de relevancia para humanos. Tradicionalmente ha habido dos medidas usadas en la valoración: memoria y precisión. Las explicaremos con la ayuda de un ejemplo. Imagínese que un sistema RD ha devuelto un conjunto resultado para una sola pregunta, para la cual sabemos qué documentos son y cuáles no son relevantes, para un corpus de 100 documentos. Las cuentas del documento en cada categoría se dan en la tabla siguiente:

	Están en conjunto resultado	No están en conjunto resultado
Relevante	30	20
No relevante	10	40

PRECISIÓN

MEMORIA

La **precisión** mide la proporción de los documentos en el conjunto resultado que son actualmente relevantes. En nuestro ejemplo, la precisión es 30/(30 + 10) = 0,75. El porcentaje de falsos positivos es 1 − 0,75 = 0,25. La **memoria** mide la proporción de todos los documentos relevantes en la colección que están en el conjunto resultado. En nuestro ejemplo, la memoria es 30/(30 + 20) = 0,60. El porcentaje de falsos negativos es 1 − 0,60 = 0,40. En una colección de documentos muy grande, tal como el World Wide Web, la memoria es difícil de calcular, porque para la relevancia no hay manera fácil de examinar cada página. Lo mejor que podemos hacer es o bien estimar la memoria muestreando o ignorarla completamente.

Un sistema puede compensar precisión contra memoria. En el extremo, un sistema que devuelve cada documento de la colección como su conjunto resultado, garantiza la memoria al 100 por cien, pero tendrá precisión baja. De una forma alternativa, un sistema puede devolver un documento sencillo y tener baja memoria, pero una precisión al 100 por cien. Una forma para resumir esta compensación es a través de una **curva ROC**. «ROC» significa «característica de funcionamiento del receptor» (lo cual no está muy claro). Es un gráfico que mide el porcentaje de falsos negativos en el eje *y* y de falsos positivos en el eje de *x*, para varios puntos de compensación. El área debajo de la curva es un resumen de la eficacia de un sistema RD.

CURVA ROC

Memoria y precisión fueron definidas cuando se hicieron las búsquedas RD sobre todo por los bibliotecarios que estaban interesados en minuciosos resultados de estudio. Hoy, la mayoría de consultas (centenares de millones por día) se hacen por los usuarios de Internet que están menos interesados en la minuciosidad y más interesados en encontrar una respuesta inmediata. Para ellos, una buena medida es la media de **graduación recíproca** del primer resultado relevante. Es decir, si un primer resultado del sistema es relevante, consigue una puntuación de 1 en la pregunta, y si los dos primeros no son relevantes, pero el tercero sí lo es, consigue una puntuación de 1/3. Una medida alternativa es el **tiempo de respuesta**, que mide cuánto tiempo lleva al usuario el hallazgo la respuesta deseada a un problema. Esto se acerca lo más posible a lo que realmente deseamos medir, pero tiene la desventaja de que cada experimento requiere actualizar los temas de prueba humanos.

GRADUACIÓN RECÍPROCA

TIEMPO DE RESPUESTA

Refinamientos RD

El modelo unigram de palabras trata todas las palabras como totalmente independientes, pero sabemos que algunas palabras están correlacionadas: «sofá» se relaciona de cerca con «sofás» y «sofá». Muchos sistemas RD procuran llevar la cuenta de estas correlaciones.

Por ejemplo, si la consulta es [sofá], no se aceptaría excluir del conjunto resultado los documentos que mencionan «SOFÁ» o « sofás» pero no «sofá». La mayoría de los sistemas RD **almacenan el caso** «SOFÁ» con «sofá» y muchos utilizan un algoritmo de **procedencia** que sirve para reducir «sofás» a la forma del origen «sofá». Típicamente esto supone un pequeño aumento en memoria (del orden del 2 por ciento para el inglés). Sin embargo, puede dañar la precisión. Por ejemplo, la procedencia de «stocking» a «stock» tenderá a disminuir la precisión para las consultas sobre zapatos o instrumen-

ALMACENAN EL CASO

PROCEDENCIA

tos financieros, aunque podría mejorar la memoria para las consultas sobre el almacenamiento. Prevenir los algoritmos basados en reglas (por ejemplo, eliminar el *ing*) no puede evitar este problema, pero pueden hacerlo los algoritmos de procedencia más recientes basados en diccionarios (no eliminar el *ing* si la palabra aparece ya en el diccionario). Mientras que la precedencia tiene un efecto pequeño en inglés, es más importante en otros idiomas. En alemán, por ejemplo, es frecuente ver palabras como «Lebensversicherungsgesellschaftsangestellter» (empleado de compañía de seguros de vida). Idiomas tales como finlandés, turco, inuit y yupik tienen reglas morfológicas recursivas que en principio generan palabras de longitud ilimitada.

SINÓNIMOS

El paso siguiente es reconocer **sinónimos**, por ejemplo «sofá» para «diván». Como con la procedencia, esto tiene el potencial para pequeños aumentos en memoria, pero con el peligro de disminuir la precisión si se aplica demasiado agresivamente. Aquellos interesados en el jugador del fútbol Antonio Silla no desearían tener resultados sobre sillas. El problema es que «los idiomas tratan los sinónimos tal como la naturaleza trata el vacío» (Cruse, 1986). Es decir, siempre que hay dos palabras que significan la misma cosa, los hablantes conspiran para modificar los significados eliminando la confusión.

CORRECCIÓN DEL DELETREO

Muchos sistemas RD utilizan **bigrams** de palabra a un cierto grado, aunque pocos ponen en ejecución un modelo bigram probabilístico completo. Se pueden utilizar las rutinas de la **corrección del deletreo** para corregir los errores en documentos y consultas.

METADATOS

Como refinamiento final, los RD pueden ser mejorados considerando datos-**metadatos** exteriores al texto del documento. Los ejemplos incluyen palabras claves y enlaces de hypertexto entre documentos proporcionados-por-los-humanos.

Presentación de los conjuntos de resultados

El principio de graduación de la probabilidad dice que un conjunto resultado se presente al usuario como lista ordenada por la probabilidad de la relevancia. Esto tiene sentido si un usuario está interesado en encontrar todos los documentos relevantes lo más rápidamente posible. Pero funciona con problemas porque no considera la *utilidad*. Por ejemplo, si hay dos copias del documento más relevante de la colección, entonces una vez que usted haya visto el primero, el segundo tiene igual importancia, pero utilidad cero. Muchos sistemas RD tienen mecanismos para eliminar los resultados que son demasiado similares a los resultados anteriores.

REALIMENTACIÓN DE LA RELEVANCIA

Una de las maneras más potentes de mejorar el funcionamiento de un sistema RD debe tener en cuenta la **realimentación de la relevancia**, realimentación del usuario diciendo qué documentos de un conjunto inicial de resultados son relevantes. El sistema puede entonces presentar un segundo conjunto de resultados con los documentos que son similares a ésos.

CLASIFICACIÓN DEL DOCUMENTO

Una aproximación alternativa es presentar el conjunto de resultados como *árbol etiquetado* en vez de una lista ordenada. Con la **clasificación del documento**, los resultados se clasifican en una taxonomía de tópicos preexistentes. Por ejemplo, una colección de historias noticias podría clasificarse en noticias del mundo, noticias locales, negocio, entretenimiento y deportes. Con el **documento *clustering***, se crea el árbol de categorí-

DOCUMENTO *CLUSTERING*

as para cada conjunto de resultados. La clasificación es apropiada cuando hay un número pequeño de asuntos en una colección, y el *clustering* es apropiado para colecciones más amplias tales como el World Wide Web. En cualquier caso, cuando el usuario realiza una consulta, el conjunto de resultados se muestra organizado en carpetas basadas en las categorías.

La clasificación es un problema de aprendizaje supervisado, y como tal, se puede atacar con cualesquiera de los métodos del Capítulo 18. Una aproximación popular es mediante árboles de decisión. Dado un sistema de entrenamiento de documentos etiquetados con las categorías correctas, podríamos construir un solo árbol de decisión en el que se asigna cada documento a la categoría apropiada. Esto funciona bien solamente cuando hay pocas categorías, pero para conjuntos de categorías más grandes construiremos un árbol de decisión para cada categoría, etiquetando el documento como un miembro o no miembro de la categoría. Generalmente, las características probadas en cada nodo son palabras individuales. Por ejemplo, un nodo en el árbol de la decisión para la categoría «deportes» podría probarse para la presencia de la palabra «baloncesto». Los árboles de decisión, los modelos sencillos de Bayes y las máquinas vectoriales han sido usados para clasificar texto; en muchos casos la exactitud está en el rango de 90-98 por ciento para clasificación booleana.

El *clustering* es un problema de aprendizaje no supervisado. En el Apartado 20.3 vimos cómo el algoritmo EM se puede utilizar para mejorar una estimación inicial de un *clustering*, basada en una mezcla del modelo Gaussiano. La tarea de *clustering* de documentos es más dura porque no sabemos qué datos fueron generados por un modelo gaussiano y porque estamos tratando un espacio de dimensión mayor. Se han desarrollado distintas aproximaciones.

CLUSTERING **AGLOMERATIVO**

El ***clustering* aglomerativo** crea un árbol de los *clusters* que van hacia abajo a documentos individuales. El árbol se puede podar en cualquier nivel para producir un número más pequeño de categorías, pero eso se considera fuera del algoritmo. Comenzamos considerando cada documento como un *cluster* separado. Después encontramos los dos racimos que están más cercanos el uno al otro según una cierta medida de la distancia y combinamos estos dos racimos en uno. Repetimos el proceso hasta que permanece un racimo. La medida de la distancia entre dos documentos es una cierta medida del solapamiento entre las palabras en los documentos. Por ejemplo, podríamos representar un documento por un vector de los contadores de palabras, y definir la distancia como la distancia euclídea entre dos vectores. La medida de la distancia entre dos *clusters* puede ser la distancia al punto medio del *cluster*, o puede ser la distancia media entre los miembros de los *clusters*. El *clustering* aglomerativo tiene un coste temporal de $O(n^2)$, donde n es el número de documentos.

CLUSTERING **K-MEDIAS**

El ***clustering* K-medias** crea un conjunto de plano exactamente k categorías. Trabaja como sigue:

1. Tomar k documentos al azar que representan las k categorías.
2. Asignar cada documento a la categoría más cercana.
3. Calcular la media de cada *cluster* y utilice las k medias para representar los nuevos valores de las k categorías.
4. Reperir los pasos (2) y (3) hasta convergencia.

K-medias tiene un coste temporal de $O(n)$, y le da ventaja sobre el *clustering* aglomerativo. Se divulga a menudo que es menos exacto que el *clustering* aglomerativo, aunque algunos autores dicen que lo puede hacer casi tan bien (Steinbach *et al.,* 2000).

Sin importar el algoritmo usado para el *clustering*, hay una tarea más antes de que el *clustering* pueda ser utilizado para presentar el conjunto resultado: encontrando una buena manera de describir el *cluster*. En la clasificación, los nombres de la categoría ya están definidos (por ejemplo, las ganancias), pero en *clustering* necesitamos inventar los nombres de las categorías. Una forma de hacerlo es elegir una lista de las palabras que son representativas del *cluster*. Otra opción es elegir el título de uno o más documentos cercanos al centro del *cluster*.

Implementar sistemas RD

Hasta ahora hemos definido cómo los sistemas RD trabajan en abstracto, pero no hemos explicado cómo hacerlos eficientes de modo que un motor de búsqueda de la Web pueda devolver los mejores resultados de una colección multibillonaria de páginas en una décima de segundo. Las dos estructuras de datos claves para cualquier sistema RD son el léxico, que enumera todas las palabras de la colección del documento, y el índice invertido, que enumera todos los lugares donde cada palabra aparece en la colección del documento.

El **léxico** es una estructura de datos que soporta una operación: dada una palabra, devuelve la localización en el índice invertido que almacena las ocurrencias de la palabra. En algunas implementaciones también devuelve el número total de los documentos que contienen la palabra. El léxico se debe implementar con una tabla *hash* o una estructura de datos similar que permita que estas operaciones de búsqueda sean rápidas. A veces un conjunto de palabras comunes con poco contenido de información será omitido del léxico, estas son las **palabras de parada** («el,» «de,» «a,» «ser,» «no,» «o,» etc.) se consideran espacio en el índice y no mejoran la puntuación de los resultados. La única buena razón de mantenerlos en el léxico es para los sistemas que soportan consultas de frase: es necesario un índice de las palabras de parada para recuperar eficientemente resultados de consultas como «ser o no ser».

PALABRAS DE PARADA

ÍNDICE INVERTIDO

LISTAS DE ENLACE

El **índice invertido**[3], como el índice de la parte posterior de este libro, consiste en un conjunto de **listas de enlace**: lugares donde ocurre cada palabra. Para el modelo booleano de la palabra clave, una lista de enlace es justo una lista de documentos. Para el modelo unigram, es una lista de pares (documento, contador). Para apoyar la búsqueda de la frase, la lista de enlace debe también incluir las posiciones dentro de cada documento donde ocurre la palabra.

Cuando la consulta es de una sola palabra (el 26 por ciento de las veces, según Silverstein *et al*. (1998)), el proceso es muy rápido. Para conseguir la dirección de la lista de éxitos hacemos búsquedas simples en el léxico, y entonces creamos una cola de prio-

[3] El término (índice invertido) es redundante; un término mejor sería justo «índice». Es invertido en el sentido de que está en diferente orden que las palabras en el texto, pero eso es como son todos los índices. Sin embargo, el índice invertido es término tradicional en RD.

ridad vacía. Después de eso, pasamos a través de la lista de éxitos un documento cada vez y comprobamos el contador para el documento. Si la cola de la prioridad tiene menos elementos que R (donde R es el tamaño deseado del conjunto resultado), agregamos el par (documento, contador) a la cola. Si no, si el contador es más grande que el de la entrada más baja en la cola de la prioridad, suprimimos la entrada más baja y agregamos el nuevo par (documento, contador). Así, contestar a la consulta tiene un coste temporal $O(H + R \log R)$, donde H es el número de documentos en la lista de éxitos. Cuando la pregunta tiene n palabras, tenemos que combinar las n listas de éxito, lo que se puede hacer con un coste $O(nH + R \log R)$.

Hemos presentado nuestra aproximación teórica de RD usando el modelo probabilístico porque ese modelo hace uso de las ideas que ya hemos utilizado para otros asuntos. Pero los sistemas actuales de RD es más probable que en la práctica utilicen una aproximación distinta llamada el **modelo de espacio vectorial**. Este modelo utiliza la misma bolsa de palabras que el modelo de la probabilidad. Cada documento se representa como vector de las frecuencias de las palabras del unigram. La consulta se representa también exactamente de la misma manera; la consulta [modelo recuperación datos Bayes] se representa como el vector

MODELO DE ESPACIO VECTORIAL

$$[0,\ldots, 1, 0,\ldots, 1, 0,\ldots, 1, 0,\ldots, 1, 0,\ldots]$$

donde la idea es que hay una dimensión para cada palabra en la colección del documento y la consulta consigue una puntuación de 0 en cada dimensión excepto las cuatro que aparecen realmente en la consulta. Los documentos relevantes son seleccionados encontrando los vectores del documento que son los vecinos más cercanos al vector de la consulta en el espacio vectorial. Una medida de la semejanza es el producto escalar entre el vector de la pregunta y el vector del documento; cuanto más grande, más cercanos están los dos vectores. Algebraicamente, esto da valores altos para las palabras que aparecen con frecuencia en el documento y en la consulta. Geométricamente, el producto escalar entre dos vectores es igual al coseno del ángulo entre los vectores; la maximización del coseno de esos dos vectores (en el mismo cuadrante) significa que el ángulo entre ellos está cerca de 0.

Hay mucho más que esto del modelo del espacio vectorial. En la práctica, ha crecido para acomodar una amplia variedad de características, refinamientos, correcciones y adiciones. La idea básica de alinear documentos por su semejanza en un espacio vectorial permite encontrar nuevas ideas en el sistema numérico graduación. Algunos argumentan que un modelo probabilístico permitiría las mismas manipulaciones y que se harían en forma más general, pero los investigadores en RD son poco proclives a cambiar a otro modelo a menos que puedan ver una ventaja clara respecto a otro modelo.

Para obtener una idea de la magnitud del problema de la indexación de direcciones para una tarea RD típica, considere una colección estándar del documento de la colección del TREC (*Text REtrieval Conference*) que consiste en 750.000 documentos que suman dos GB (gigabytes) de texto. El léxico contiene 500.000 palabras después de procedencia y plegamiento del caso; estas palabras se pueden almacenar en un número de siete a l0 MB. El índice invertido con pares (documento, contador) usa 324 MB, aunque se pueden utilizar técnicas de compresión para conseguir reducirlo a 83 MB. La compresión ahorra espacio, aumentado levemente el coste de procesamiento. Sin embargo,

si la compresión permite que se mantenga el índice entero en memoria en vez de almacenarlo en disco, después proporcionará un aumento sustancial neto en rendimiento. La ayuda para las consultas de frase aumenta el tamaño a cerca de 1.200 MB sin comprimir o 600 MB con compresión. Los motores de búsqueda de la Web funcionan en una escala alrededor de 3.000 veces más grande que esto. Muchos de los beneficios son los mismos, pero por ser impracticable tratar con terabytes de datos sobre un solo computador, el índice se divide en k segmentos, con cada segmento almacenado en un computador distinto. Se envía en paralelo una pregunta a todos los computadores, y entonces los k conjuntos de resultados se combinan en un solo conjunto resultado que se muestra al usuario. Los motores de búsqueda de la Web también tienen que ocuparse de millares de consultas por segundo, así que necesitan n copias de los k computadores. Los valores de k y de n continúan creciendo con el tiempo.

23.3 Extracción de la información

EXTRACCIÓN DE LA INFORMACIÓN

La **extracción de la información** es el proceso de crear entradas de la base de datos a partir de un texto y buscar ocurrencias de una clase particular de objeto o acontecimiento para las relaciones entre esos objetos y acontecimientos. Podríamos intentar extraer instancias de direcciones de las páginas de la Web, con campos de la base de datos para la calle, la ciudad, el estado y el código postal; o los casos de tormentas en informes del tiempo, con campos para la temperatura, velocidad del viento, y precipitación. Los sistemas de extracción de información se sitúan a mitad de camino entre los sistemas de la recuperación de datos y los programas de análisis completo del texto, en los que necesitan más que un documento como bolsa de palabras, pero menos que analizar totalmente cada frase.

El tipo más simple de sistema de extracción de la información se llama un sistema basado en atributos porque asume que el texto entero se refiere a un solo objeto y la tarea es extraer cualidades de ese objeto. Por ejemplo, mencionamos en el Apartado 10.5 el problema de extraer del texto «monitor de 17 pulgadas SXGA por solamente 249,99 dólares» las relaciones de la base de datos vienen dadas por

$$\exists m \quad m \in ComputerMonitors \wedge Size(m, pulgadas(17)) \wedge Precio(m, \$(249{,}99)) \wedge Resolución(m, 1280 \times 1024)$$

EXPRESIONES REGULARES

Parte de esta información se puede manejar con la ayuda de **expresiones regulares**, que definen una gramática regular en una sola secuencia de texto. Las expresiones regulares se utilizan en comandos de Unix tales como grep, en lenguajes de programación como Perl y en procesadores de textos como Microsoft Word. Los detalles varían levemente de una herramienta a otra y se aprenden mejor en el manual apropiado, pero aquí mostramos cómo construir una expresión regular para los precios en dólares, mostrando subexpresiones comunes:

`[0-9]`	empareja cualquier dígito de 0 a 9
`[0-9]+`	empareja uno o más dígitos
`.[0-9][0-9]`	empareja un punto decimal seguido por dos dígitos

`(.[0-9] [0-9])?`	empareja un punto decimal seguido por dos dígitos, o nada
`$[0-9]+(.[0-9][0-9])?`	empareja 249,99 dólares o 1,23 dólares o 1.000.000 dólares o…

Los sistemas de extracción basada en atributos pueden ser construidos como series de expresiones regulares, una para cada atributo. Si una expresión regular empareja el texto exactamente una vez, entonces podemos sacar la porción del texto que es el valor del atributo. Si no hay emparejamiento no hay nada más que podamos hacer, pero si hay varios, necesitamos un proceso para elegir entre ellos. Una estrategia es tener varias expresiones regulares para cada atributo, ordenadas por prioridad. Así pues, por ejemplo, la expresión regular de prioridad superior para el precio podría buscar la secuencia «nuestro precio» inmediatamente antes del signo dólar; si no se encuentra, caemos en una expresión regular menos fiable. Otra estrategia es tomar todos los emparejamientos y encontrar alguna manera para elegir entre ellos. Por ejemplo, podríamos tomar el precio más bajo que está dentro del 50 por ciento del precio más alto. Esto manejará los textos como «el precio de lista 99,00 dólares, el precio de venta especial 78,00 dólares, transporte 3,00 dólares».

Un avance de los sistemas de la extracción basada en atributos son los sistemas de extracción relacionales, que tienen que preocuparse de más de un objeto y de las relaciones entre ellos. Así, cuando estos sistemas consideran el texto «249,49 dólares», necesitan determinar no sólo que es un precio, sino también qué objeto tiene ese precio. Un sistema de extracción relacional típico es FASTUS, que maneja historias de noticias sobre fusiones y adquisiciones corporativas. Puede leer la historia:

> Bridgestone Sports Co. dijo el viernes que se ha instalado una empresa de riesgo compartido en Taiwán con una agrupación local y una casa japonesa para producir clubs de golf que se enviarán a Japón.

y generar un registro de la base de datos como

$$e \in EmpresaDeRiesgoCompartido \wedge Producto(e, «palo\ de\ golf») \wedge Fecha(e, «Viernes») \wedge Entidad(e, «Bridgestone\ Sports\ Co») \wedge Entidad(e, «una\ agrupación\ local») \wedge Entidad(e, «una\ casa\ japonesa»).$$

TRANSDUCTORES DE ESTADO FINITO CONECTADOS EN CASCADA

Los sistemas de extracción relacional se construyen a menudo usando **transductores de estado finito conectados en cascada**. Es decir, consisten en una serie de autómatas de estados finitos (FSAs), donde cada autómata recibe el texto como entrada, transforma el texto en un formato diferente, y lo pasa al siguiente autómata. Esto es apropiado porque cada FSA puede ser eficiente y porque juntos pueden extraer la información necesaria. Un sistema típico es FASTUS, que consiste en las cinco etapas siguientes:

1. Troceado
2. Procesamiento de palabras complejas
3. Procesamiento de grupos básicos
4. Procesamiento de frases complejas
5. Combinación de estructuras

La primera etapa de FASTUS es el **troceado**, que divide la entrada en segmentos de caracteres formando símbolos (palabras, número y signos de puntuación). Para el inglés, el troceado puede ser bastante simple: los caracteres de separación son el espacio en blanco o los signos de puntuación permitiendo un trabajo bastante bueno. Para el japonés,

el troceado necesitaría realizar la segmentación, usando algo como el algoritmo de segmentación de Viterbi (*véase* la Figura 23.1). Algunos troceadores tienen que ver con lenguajes de marcado tales como HTML, SGML y XML.

La segunda etapa **procesamiento de palabras complejas**, incluyendo colocaciones tales como «empresa de riesgo compartido» así como nombres propios por ejemplo «primer ministro Tony Blair» y «Bridgestone Sports Co.». Éstos son reconocidos por una combinación de entradas léxicas y de reglas gramáticales de estado finito. Por ejemplo, un nombre de compañía se podría reconocer por la regla

PalabraMayúscula + («Compañía» | «Co» | «s.a.»)

Estas reglas se deben construir con cuidado y probar para memoria y precisión. Un sistema comercial reconoció «presidente de Intel Andy Arboleda» como un lugar en vez de una persona debido a una regla de la forma

PalabraMayúscula + («Arboleda» | «Bosque» | «Ciudad» | …)

La tercera etapa procesa **grupos básicos**, entendiendo como tales los grupos de sustantivos y los grupos de verbos. La idea es poner los trozos de estos grupos en las unidades que serán manejadas por las fases posteriores. Un grupo de sustantivo consiste en un sustantivo principal, precedido opcionalmente por determinantes y otros modificadores. Debido a que el grupo de sustantivo no incluye la complejidad completa del *FN* en ε_1 no necesitamos reglas recursivas de la gramática independiente del contexto: las reglas de la gramática regular permitidas en autómatas de estados finitos son suficientes. El grupo del verbo consiste en un verbo y sus auxiliares y adverbios asociados, pero sin los objetos directo e indirecto y las preposiciones. La frase del ejemplo saldría de esta etapa como sigue:

```
 1 NG: Bridgestone Sports Co.            | 10 NG: una agrupación loca
 2 VG: dijo                              | 11 CJ: y
 3 NG: el viernes                        | 12 NG: una casa japonesa
 4 NG: se                                | 13 VG: para producir
 5 VG: ha instalado                      | 14 NG: palos de golf
 6 NG: una empresa de riesgo compartido  | 15 VG: que se enviarán
 7 PR: en                                | 16 PR: a
 8 NG: Taiwan                            | 17 NG: Japón
 9 PR: con
```

Aquí NG significa grupo de sustantivo, VG es grupo del verbo, PR es preposición, y CJ es conjunción.

La cuarta etapa combina los grupos básicos en **frases complejas**. De nuevo, el objetivo es tener reglas que sean de estado finito y se puedan procesar así rápidamente, y que den lugar a frases inequívocas (o casi inequívocas). Un tipo de regla de combinación trata de acontecimientos específicos del dominio. Por ejemplo, la regla

Compañía + InstaladoEmpresaDeRiesgoCompartido («con» Compañía+)?

captura una manera de describir la formación de una empresa de riesgo compartido. Esta etapa es la primera en la cascada donde la salida se pone en una plantilla de la base de datos y también se coloca en la secuencia de salida.

La etapa final es la **combinación de estructuras** que fueron acumuladas en el paso anterior. Si la frase siguiente dice «la empresa de riesgo compartido comenzará la producción en enero» entonces este paso determinará que hay dos referencias a riesgo compartido, y que deben ser combinadas en una.

En general, la extracción de la información funciona bien para dominios restringidos en los cuales sea posible predeterminar qué temas serán discutidos, y cómo serán mencionados. Se ha probado útilmente en ciertos dominios, pero no sustituye al procesamiento del lenguaje natural a escala completa.

23.4 Traducción automática

La traducción automática es la traducción automática de un texto en un lenguaje natural (la fuente) a otro (el objetivo, el destino final). Este proceso se ha demostrado que es útil para un número de tareas, incluyendo las siguientes:

1. **Traducción literal,** en la que el objetivo es simplemente conseguir tomar la idea central de un trozo de texto. Las frases son gramaticalmente incorrectas y poco elegantes, pero se toleran con tal de que el significado sea claro. Por ejemplo, cuando se navega en la Web, un usuario a menudo queda satisfecho con una traducción literal de una página en una lengua extranjera. A veces una persona monolingüe puede editar la salida sin tener que leer la fuente. Este tipo de traducción automatizada ahorra dinero porque a los editores se les puede pagar menos que a los traductores bilingües.
2. **Traducción de fuente-restringida**, en donde la materia del asunto y el formato del texto original están severamente limitados. Uno de los ejemplos más acertados es el sistema TAUM-METEO, que traduce informes metereológicos del inglés al francés. Trabaja porque el lenguaje que se utiliza en este tipo de informes tiene un gran estilo y es regular.
3. **Traducción preeditada,** en donde una persona preedita el documento fuente para hacerlo concordar con un subconjunto restringido del inglés (o de cualquiera que sea la lengua original) antes de la traducción automática. Este estudio es particularmente rentable cuando hay necesidad de traducir un documento a varios idiomas, como es el caso de los documentos jurídicos en la Comunidad Europea o para empresas que venden el mismo producto en muchos países. Los lenguajes restringidos a veces son llamados «Inglés Caterpillar» porque la empresa Caterpillar Corp. fue la primera empresa en intentar escribir sus manuales de esta manera. Xerox definió un lenguaje para sus manuales de mantenimiento lo suficientemente simple para que pudiese ser traducido automáticamente a todos los idiomas a los que Xerox necesitaba que fueran traducidos. Como una ventaja añadida, los manuales originales en inglés se hicieron también más claros.
4. **Traducción literaria**, donde todos los matices del texto original se preservan. Esto va más allá de lo actualmente cubierto en el estado del arte de la traducción automática.

Como ejemplo de traducción literal, el sistema de traducción SYSTRAN tradujo el primer párrafo de este capítulo, del italiano al inglés, de la siguiente forma:

Italian: In capitolo 22 abbiamo visto come un agente potrebbe comunicare con un altro agente (essere umano o software) che usando le espressioni in un linguaggio reciprocamente accordato. Completare sintattico e l'analisi semantica delle espressioniè necessaria da estrarre il significato completo del utterances ed è possibile perché le espressioni sono corte e limitate ad un settore limitato.

English: In chapter 22 we have seen as an agent could communicate with an other agent (to be human or software) that using the expressions in a language mutual come to an agreement. Complete syntactic and the semantic analysis of the expressions is necessary to extract the complete meant one of the utterances and is possible because the expressions short and are limited to a dominion.

La traducción es difícil porque, en un caso general, requiere un conocimiento en profundidad del texto, y éste a su vez requiere un conocimiento en profundidad de la situación que se comunica. Esto es así incluso para textos muy simples, incluso textos de una sola palabra. Consideremos la palabra «Open» en la puerta de un almacén[4]. Comunica la idea de que el almacén está aceptando clientes en este momento. Consideremos de nuevo la misma palabra «Open» en un gran cartel fuera de un almacén construido recientemente. Significa que el almacén está ahora abierto diariamente, pero los que lean esta palabra no se sentirán engañados si el almacén cierra por la noche sin quitar el cartel. Los dos rótulos usan la misma palabra con diferente significado. En Alemania, sin embargo, la palabra en la puerta puede ser «Offen» mientras que en el cartel se leerá «Neu Eröffnet».

El problema es que idiomas diferentes categorizan el mundo de manera diferente. Por ejemplo, la palabra francesa «doux» recoge un amplio rango de significados correspondiente aproximadamente a las palabras inglesas «soft», «sweet», y «gentle». De forma similar, la palabra inglesa «hard» recoge virtualmente todos los usos de la palabra alemana «hart» (físicamente recalcitrante, cruel) y algunos usos de la palabra «schwierig» (difícil). El verbo alemán «heilen» recoge los usos médicos de la palabra inglesa «cure», además de los usos transitivos e intransitivos de «heal». Por tanto, representar el significado de una frase es más difícil para la traducción que la comprensión de un lenguaje sencillo. Un sistema de análisis de un lenguaje sencillo puede usar predicados como $Open(x)$, pero para la traducción, el lenguaje de representación tendría que tener más distinciones, por ejemplo con $Open_1(x)$ representando el sentido «Offen» y $Open_2(x)$ representando el sentido «Neu Eröffnet». Se llama **interlingua** a un lenguaje de representación que haga todas las distinciones necesarias para un conjunto de lenguajes.

INTERLINGUA

Para hacer una traducción fluida, un traductor (humano o máquina) debe leer el texto original, comprender la situación a la cual se está refiriendo, y encontrar el texto correspondiente en el lenguaje destino que dé como resultado un buen trabajo de describir lo mismo o una situación similar. A menudo, esto requiere una elección. Por ejemplo, la palabra inglesa «you», cuando se refiere a una sola persona se puede traducir al francés tanto por la palabra formal «vous» como por la informal «tu». No hay forma de re-

[4] Este ejemplo es debido a Martin Kay.

ferirse al concepto de «you» en francés sin hacer una elección de la manera formal o informal. Los traductores (tanto máquinas como humanos) a veces encuentran dificultades para hacer esta elección.

Sistemas de traducción automáticos

Los sistemas de traducción automáticos varían en el nivel con el que analizan el texto. Algunos sistemas intentan analizar el texto de entrada de principio a fin en una representación interlingua (como hicimos en el Capítulo 22) y entonces generan frases en el lenguaje de destino a partir de esa representación. Esto es difícil porque incluye el problema de la comprensión completa del lenguaje como un subproblema, a lo que se añade la dificultad de tratar con una interlingua. Es un sistema frágil porque si el análisis falla no se produce ningún resultado. Tiene la ventaja de que no hay ninguna parte del sistema que dependa del conocimiento de los dos idiomas a la vez. Esto significa que uno puede construir un sistema de interlingua para traducir n lenguajes con una complejidad de $O(n)$ en vez de $O(n^2)$.

TRANSFERENCIA

Otros sistemas se basan en una **transferencia**. Mantienen una base de datos de las reglas de traducción (o ejemplos) y en el momento en que la regla (o un ejemplo) se corresponde, lo traducen directamente. La transferencia puede darse a nivel semántico, sintáctico o léxico. Por ejemplo, una regla sintáctica estricta cambia el orden del inglés [*Adjetivo Nombre*] al francés [*Nombre Adjetivo*]. Una regla mixta, sintáctica y léxica, cambia el francés [S_1 «et puis» S_2] al inglés [S_1 «and then» S_2]. A una transferencia que va directamente de una frase a otra se le llama método de **traducción basado en la memoria**, porque se basa en memorizar un gran conjunto de pares (inglés, francés). El método de transferencia es robusto porque siempre genera *alguna* salida y, al menos, alguna de las palabras tiene que ser correcta. La Figura 23.5 representa los diferentes puntos de transferencia.

TRADUCCIÓN BASADA EN LA MEMORIA

Figura 23.5 Diagrama esquemático de las decisiones para un sistema de traducción. Comenzamos por el texto en inglés en la parte superior. Un sistema basado en interlingua sigue las líneas continuas, analizando el inglés primero en forma sintáctica, después en una representación semántica y una representación interlingua y entonces a través de la generación de una forma semántica, sintáctica y léxica en francés. Un sistema basado en transferencia utiliza las líneas punteadas como un atajo. Sistemas diferentes hacen la transferencia en puntos diferentes; algunos pueden hacerla en múltiples puntos.

Traducción automática estadística

A principios de los 60, se tuvo la gran esperanza de que los computadores fueran capaces de traducir de un lenguaje natural a otro tal y como el proyecto de Turing tradujo mensajes codificados en alemán a alemán comprensible. En 1966, parecía claro que una traducción fluida requería una comprensión del significado del mensaje, mientras que descodificar el código no.

En la década pasada, hubo un cierto impulso hacia los sistemas de traducción automática basados en métodos estadísticos. Por supuesto, cualquiera de los pasos descritos en la Figura 23.5 se puede beneficiar de la aplicación de datos estadísticos y de un modelo probabilístico claro de aquello que constituye un buen análisis o transferencia. Pero la «traducción automática estadística» ha venido a denotar una aproximación al problema de traducción completo, basada en encontrar la traducción más probable de una frase, usando datos recopilados de un corpus bilingüe. Como ejemplo de corpus bilingüe, **Hansard**[5] es una grabación de un debate parlamentario. Canadá, Hong Kong y otros países utilizan Hansards bilingües, la Unión Europea publica sus documentos oficiales en 11 idiomas, y las Naciones Unidas publican documentos de forma «multilingüe». Se ha demostrado que son recursos de gran valor para la traducción automática estadística.

HANSARD

Podemos expresar el problema de traducir una frase en inglés E, por ejemplo, a una frase en francés[6] F mediante la siguiente aplicación de la regla de Bayes:

$$\begin{aligned}\text{argmax}_F\, P(F \mid E) &= \text{argmax}_F\, P(E \mid F)P(F)/P(E)\\ &= \text{argmax}_F\, P(E \mid F)P(F)\end{aligned}$$

Esta regla dice que deberíamos considerar todas las posibles frases en francés F y elegir aquella que maximiza el producto $P(E \mid F)P(F)$. El factor $P(E)$ puede ser ignorado porque es el mismo para cualquier F. El factor $P(F)$ es el **modelo de lenguaje** para el francés; dice qué probabilidad hay de que una frase dada esté en francés. $P(E \mid F)$ es el **modelo de traducción**; dice qué probabilidad hay de que una frase en inglés sea una traducción de una determinada frase en francés.

MODELO DE LENGUAJE

MODELO DE TRADUCCIÓN

El lector astuto se preguntará qué hemos ganado al definir $P(F \mid E)$ en términos de $P(E \mid F)$. En otras aplicaciones de la regla de Bayes hicimos lo inverso porque queríamos usar un modelo causal. Por ejemplo, usamos el modelo causal $P(\textit{Síntomas}/\textit{Enfermedad})$ para calcular $P(\textit{Enfermedad} \mid \textit{Síntomas})$. Con la traducción, en cambio, ninguna de las dos direcciones es más causal que la otra. La razón de aplicar la regla de Bayes en este caso es que creemos que podemos aprender un modelo de lenguaje $P(F)$ que es más preciso que el modelo de traducción $P(E \mid F)$ (y más preciso que estimar $P(F \mid E)$ directamente). Esencialmente, hemos dividido el problema en dos partes: primero utilizamos el modelo de traducción $P(F \mid E)$ para encontrar frases francesas candidatas que

[5] Llamado después Guillermo Hansard, quien publicó por primera vez los debates parlamentarios británicos en 1811.

[6] A lo largo de esta apartado consideramos el problema de traducir del inglés al francés. No debemos confundirnos por el hecho de que la regla de Bayes nos conduce a considerar $P(E \mid F)$ en vez de $P(F \mid E)$, dando la impresión de que traducimos del francés al inglés.

mencionen los conceptos correctos de la frase en inglés, pero que podría no ser un francés fluido; entonces usamos el modelo de lenguaje $P(F)$ (para el cual tenemos estimaciones de probabilidad mucho mejores) para elegir al mejor candidato.

El **modelo de lenguaje** $P(F)$ puede ser cualquier modelo que dé una probabilidad para una frase. Con un corpus muy grande podemos estimar $P(F)$ directamente contando cuántas veces aparece cada frase en el corpus. Por ejemplo, si usamos la Web para recoger 100 millones de frases en francés, y si la frase «Clique ici» aparece 50.000 veces, entonces P(Clique ici) es .0005. Pero incluso con 100 millones de ejemplos, muchas frases no aparecerán ninguna vez[7]. Por tanto, utilizaremos el modelo familiar de lenguaje bigram, en el que la probabilidad de una frase en francés consistente en las palabras $f_1 \ldots f_n$ es

$$P(f_1 \cdots f_n) = \prod_{i=1}^{n} P(f_i \mid f_{i-1})$$

Necesitamos conocer las probabilidades bigram tales como $P(\text{Eiffel} \mid \text{tour}) = .02$. Esto recoge sólo una noción muy local de la sintaxis, donde una palabra depende justo de la palabra anterior. Sin embargo, para una traducción literal es a menudo suficiente[8].

El **modelo de traducción** $P(E \mid F)$ es más difícil de conseguir. Primero, no tenemos preparada una colección de pares de frases (inglés, francés) con las que entrenar. Y además, la complejidad del modelo es mayor ya que considera el producto cruzado de frases, más que las frases individuales. Comenzaremos con un modelo de traducción demasiado simplista y construiremos algo aproximado al «IBM Model 3» (Brown *et al.*, 1993) que podría parecer todavía demasiado simple, pero que ha demostrado generar traducciones literales aceptables en la mitad de tiempo.

El modelo demasiado simplista es «traducir una frase, sólo traducir cada palabra individual e independientemente, en orden de izquierda a derecha». Este es un modelo de elección de palabra unigram. Esto hace más fácil calcular la probabilidad de una traducción:

$$P(E \mid F) = \prod_{i=1}^{n} P(E_i \mid F_i)$$

En pocas ocasiones este modelo trabaja bien. Por ejemplo, consideremos

$$P(\text{the dog} \mid \text{le chien}) = P(\text{the} \mid \text{le}) \times P(\text{dog} \mid \text{chien})$$

Bajo cualquier conjunto razonable de valores de probabilidad, «the dog» tendría la máxima probabilidad de traducción por «le chien». En la mayoría de los casos, sin embargo, el modelo falla. Un problema es el orden de las palabras. En francés «dog» es «chien» y «brown» es «brun» pero «brown dog» es «chien brun». Otro problema es que

[7] Si hay sólo 100.000 palabras en el léxico, el 99,99999 por ciento de las frases con tres palabras tienen un cero la cuenta de 100 millones de frases del corpus. El resultado es peor para frases más largas.

[8] Para los puntos más finos de la traducción $P(f_i \mid f_{i-1})$ claramente no es bastante. Por ejemplo, Marcel Proust en la página 3500 de la novela *A la récherche du temps perdu* comienza y termina con la misma palabra, así que algunos traductores han decidido hacer lo mismo, basando la traducción de una palabra en una que apareció unos dos millones de palabras anteriormente.

la elección de palabras no tiene por qué ser una correspondencia uno a uno con cada palabra, es decir, por ejemplo la palabra inglesa «home» a menudo se traduce por «à la maison», como una correspondencia de uno a tres (o de tres a uno en la otra dirección). A pesar de estos problemas, el «IBM Model 3» persistentemente sigue básicamente un modelo unigram, pero añade algunas complicaciones para mejorarlo.

FERTILIDAD

Para manejar el hecho de que las palabras no se pueden traducir una a una, el modelo introduce la noción de **fertilidad** de una palabra. Una palabra con fertilidad n consigue ser copiada hasta n veces, y entonces cada una de estas n copias se puede traducir independientemente. El modelo contiene parámetros para $P(Fertilidad = n \mid palabra)$ para cada palabra en francés. Para traducir «à la maison» como «home» el modelo elegiría fertilidad 0 para «à» y «la» y fertilidad 1 para «maison» y entonces utilizar el modelo de traducción unigram para traducir «maison» como «home». Esto parece suficientemente razonable: «à» y «la» son palabras de bajo contenido que pueden razonablemente ser traducidas como nada. Traducir en la otra dirección es más incierto. A la palabra «home» se le asignaría fertilidad 3, y da como resultado «home home home». El primer «home» se tiene que traducir como «à» el segundo como «la» y el tercero como «maison». En términos del modelo de traducción, «à la maison» tendría la misma probabilidad que «maison la à». (Esta es la parte dudosa.) Sería tarea del modelo de lenguaje decidir cuál es mejor. Puede parecer que tiene más sentido traducir directamente «home» a «à la maison» que hacerlo indirectamente vía «home home home» pero eso requeriría muchos más parámetros, y sería mucho más difícil de obtener a partir del corpus disponible.

La parte final del modelo de traducción es permutar las palabras a las posiciones correctas. Esto se hace mediante un modelo de compensación por el cual una palabra se mueve desde su posición original a su posición final. Por ejemplo, al traducir «chien brun» como «brown dog», la palabra «brown» tiene una compensación de $+1$ (se mueve una posición a la derecha) y «dog» tiene una compensación de -1. Podemos imaginar que la compensación será dependiente de la palabra: los adjetivos como «brown» tenderán a tener una compensación positiva porque el francés tiende a poner los adjetivos después del sustantivo. Pero el «IBM Model 3» decide que hacer compensaciones dependientes de la palabra requeriría demasiados parámetros, por lo tanto la compensación es independiente de la palabra y depende sólo de la posición dentro de la frase, y de la longitud de las frases en ambos idiomas. Es decir, el modelo estima los parámetros

$$P(Compensación = o \mid Posición = p, LenIng = m, LenFr = n)$$

Por tanto, para determinar la compensación para «brown» dentro de «brown dog», consultamos $P(Compensación \mid 1,2,2)$, que podría darnos, por ejemplo, $+1$ con probabilidad .3 y 0 con probabilidad .7. El modelo de compensación parece incluso más dudoso, como si se hubiera confeccionado por alguien más familiarizado con mover palabras magnéticas alrededor de un frigorífico que con hablar en la actualidad un lenguaje natural. Veremos en seguida que fue diseñado de esa manera, no porque fuera un buen modelo de lenguaje, sino porque hacía un uso razonable de los datos disponibles. En cualquier caso, debemos tener muy en cuenta que un modelo de traducción mediocre puede con-

seguirse a partir de un buen modelo del idioma francés. Aquí hay un ejemplo que muestra todos los pasos en la traducción de una frase:

Fuente en francés:	Le	chien	brun	n'	est	pas	allé	à	la	maison
Modelo de fertilidad:	1	1	1	1	1	0	1	0	0	1
Transformación en francés:	Le	chien	brun	n'	est		allé			maison
Modelo de elección de palabra:	The	dog	brown	not	did		go			home
Modelo de compensación:	0	+1	−1	+1	−1		0			0
Resultado en inglés:	The	brown	dog	did	not		go			home

Ahora, sabemos cómo calcular la probabilidad $P(F \mid E)$ para cualquier par de frases (francés, inglés). Pero lo que realmente queremos hacer es, dada una frase en inglés, encontrar la frase en francés que maximiza esa probabilidad. No podemos enumerar simplemente las frases; con 10^5 palabras en francés, hay 10^{5n} frases de longitud n, y muchas combinaciones para cada una. Incluso si consideramos solamente las 10 traducciones palabra-por-palabra más frecuentes para cada palabra, y sólo consideramos compensaciones de 0 o ± 1, obtendríamos alrededor de $2^{n/2}10^n$ frases, lo que significa que podríamos enumerarlas todas para $n = 5$, pero no para $n = 10$. En cambio, necesitamos buscar la mejor solución. Una versión de la búsqueda de A^* que se ha comprobado que es efectiva puede verse en Germann *et al*. (2001).

Probabilidades de aprendizaje para la traducción automática

Hemos diseñado un modelo para $P(F \mid E)$ que implica cuatro conjuntos de parámetros:

Modelo de lenguaje: $P(palabra_i \mid palabra_{i-1})$
Modelo de fertilidad: $P(Fertilidad = n \mid palabra_F)$
Modelo de elección de palabra: $P(palabra_E \mid palabra_F)$
Modelo de compensación: $P(Compensación = o \mid pos, len_E, len_F)$

Incluso con un vocabulario modesto de 1.000 palabras, este modelo requiere millones de parámetros. Obviamente, tenemos que aprenderlas a partir de los datos. Asumiremos que los únicos datos disponibles para nosotros son un corpus bilingüe. A continuación se muestra cómo usarlo:

Segmentar en frases: la unidad de traducción es una frase, así que tendremos que dividir el corpus en frases. Los puntos son indicadores evidentes del final de una frase, pero consideremos «el Dr. J. R. Smith de Rodeo Dr. llegó»; sólo el punto final indica final de frase. La segmentación de frases puede ser hecha con aproximadamente un 98 por ciento de precisión.

Estimar el modelo del lenguaje francés $P(palabra_i \mid palabra_{i-1})$: considerando sólo la mitad francesa del corpus, contar las frecuencias de pares de palabras y entonces hacer un promedio para dar una estimación de $P(palabra_i \mid palabra_{i-1})$. Por ejemplo, podríamos tener $P(\text{Eiffel} \mid \text{tour}) = .02$.

Alinear frases: para cada frase en la versión inglesa, determinar qué frase(s) se corresponden con la versión francesa. Normalmente, la siguiente frase de inglés corresponde a la siguiente frase de francés con una correspondencia 1:1, pero a veces hay alguna variación: una frase en un lenguaje se dividirá en una correspondencia 2:1, o el orden de dos frases se intercambiará, resultando una correspondencia 2:2. Mirando únicamente las longitudes de las frases, es posible alinearlas (1:1, 1:2, o 2:2, etc.) con una precisión entre el 90 y el 99 por ciento usando una variación del algoritmo de segmentación de Viterbi (Figura 23.1). Incluso se puede conseguir una mejor alineación utilizando marcadores que son comunes en ambos lenguajes, tales como números o nombres propios, o palabras que sabemos que tienen una traducción que no es ambigua a partir de un diccionario bilingüe.

Ahora estamos preparados para estimar los parámetros del modelo de traducción. Lo haremos primero haciendo una estimación inicial pobre y después mejorándola.

Estimar el modelo de fertilidad inicial $P(Fertilidad = n \mid palabra_F)$: dada una frase en francés de longitud m que está alineada con una frase en inglés de longitud n, consideramos esto como una evidencia de que cada palabra en francés tiene una fertilidad n/m. Considerar todas las evidencias sobre todas las frases para conseguir una distribución de probabilidad de la fertilidad para cada palabra.

Estimar el modelo de elección de palabra inicial $P(palabra_E \mid palabra_F)$: mirar todas las frases francesas que contienen, por ejemplo «brun». Las palabras que aparecen más frecuentemente en las frases del inglés alineadas con estas frases son las probables traducciones palabra a palabra de «brun».

Estimar el modelo de compensación inicial $P(Compensación = o \mid pos, len_E, len_F)$: ahora que tenemos el modelo de elección de palabra, lo usamos para estimar el modelo de compensación. Para una frase en inglés de longitud n que está alineada con una frase en francés de longitud m, miramos cada palabra francesa en la frase (en la posición i) y cada palabra inglesa en la frase (en la posición j) que es una posible elección de palabra para la palabra francesa, y lo consideramos como una evidencia para $P(Compensación = i\text{-}j \mid i, n, m)$.

VECTOR DE ALINEAMIENTO DE PALABRAS

Mejorar todas las estimaciones: utilizamos EM (esperanza-maximización) para mejorar las estimaciones. La variable oculta será un **vector de alineamiento de palabras** entre pares de frases alineadas. El vector da, para cada palabra inglesa, la posición en la frase francesa de la correspondiente palabra francesa. Por ejemplo, podríamos tener lo siguiente:

Fuente en francés:	Le	chien	brun	n’	est	pas	allé	à	la	maison
Resultado en inglés:	The	brown	dog	did	not		go			home
Alineamiento de palabras:	1	3	2	5	4		7			10

Primero, usando las estimaciones actuales de los parámetros, creamos un vector de alineamiento de palabra para cada par de frases. Esto nos permitirá tener mejores estimaciones. El modelo de fertilidad se estima contando cuántas veces un miembro del vector de alineamiento de palabra se corresponde a múltiples palabras o a cero palabras. El modelo de elección de palabra puede ahora mirar solamente las palabras que están alineadas a otras, más que mirar todas las palabras de la frase, y el modelo de compensación puede entonces mirar a cada posición de la frase para ver cuántas veces se mueve de acuer-

do con el vector de alineamiento de palabra. Desgraciadamente, no podemos saber con seguridad cuál es el alineamiento correcto, y entonces hay muchos de ellos que se pueden enumerar. Por tanto, estamos forzados a buscar unos cuantos alineamientos de alta probabilidad y darles pesos a partir de sus probabilidades cuando recogemos la evidencia para nuevas estimaciones de parámetros. Esto es todo lo que necesitamos para el algoritmo EM. A partir de parámetros iniciales calculamos los alineamientos y a partir de estos alineamientos mejoramos las estimaciones de los parámetros. Repetimos esto hasta la convergencia.

23.5 Resumen

Los principales puntos de este capítulo son:

1. Los modelos del lenguaje probabilísticos basados en *n*-grams recogen una cantidad sorprendente de información sobre el lenguaje.
2. Los CFGs pueden ampliarse a CFGs probabilísticos, haciendo más fácil el aprendizaje a partir de datos y más fácil también la no ambigüedad.
3. Los sistemas de **recuperación de información** usan un modelo del lenguaje muy simple basado en conjuntos de palabras, e intentan todavía manejarlos en términos de **memoria** y **precisión** en grandes trozos de texto.
4. Los sistemas de **extracción de información** usan un modelo más complejo que incluye nociones limitadas de sintaxis y semántica. A menudo se implementan usando una cascada de autómatas de estados finitos.
5. Los sistemas de **traducción automática** se han implementado usando una gama de técnicas, desde el análisis sintáctico y/o semántico completo hasta técnicas estadísticas basadas en frecuencias de palabras.
6. Para construir un sistema estadístico del lenguaje, lo mejor es desarrollar un modelo que pueda hacer un buen uso de los datos disponibles, incluso si el modelo parece demasiado simple.

Notas bibliográficas e históricas

Los modelos de carta *n*-gram para el modelado del lenguaje fueron propuestos por Markov (1913). Claude Shannon (Shannon y Weaver, 1949) fue el primero en generar modelos de palabra *n*-gram del inglés. Chomsky (1956, 1957) precisó las limitaciones de los modelos de estados finitos comparados con los modelos de contexto libre, concluyendo que «los modelos probabilísticos no toman en consideración algunos de los problemas básicos de la estructura sintáctica». Esto es cierto, pero ignora el hecho de que los modelos probabilísticos se introducen en algunos otros problemas básicos, problemas que los CFGs no consideran. Las conclusiones de Chomsky tuvieron el desafortunado efecto de alejar a mucha gente del modelo estadístico durante dos décadas, hasta que estos modelos volvieron a surgir para su uso en reconocimiento del habla (Jelinek, 1976).

El suavizado suma-uno es debido a Jeffreys (1948), y el suavizado de interpolación eliminada es debido a Jelinek y a Mercer (1980), que lo utilizaron para el reconocimiento del habla. Otras técnicas incluyen el suavizado Witten-Bell (1991) y el suavizado Good-Turing (Church y Gale, 1991). Este último también aparece frecuentemente en problemas de bioinformática. La bioestadística y la probabilidad NLP están acercándose, ya que trabajan con grandes secuencias estructuradas escogidas de un alfabeto de componentes.

La carta simple *n*-gram y los modelos de palabra no son los únicos modelos probabilísticos posibles. Blei *et al*. (2001) describen un modelo de texto probabilístico llamado **asignación latente de Dirichlet** que ve un documento como una mezcla de tópicos, cada uno con su propia distribución de palabras. Este modelo se puede considerar como una extensión y racionalización del modelo **indexado semántico latente** (Deerwester *et al*., 1990) (*véase* también Papadimitriou *et al*. (1998)) y está también relacionado con el modelo de mezcla de causa múltiple de (Sahami *et al*., 1996).

Las gramáticas probabilísticas independientes del contexto (PCFGs) responden a todas las objeciones de Chomsky sobre los modelos probabilísticos, y presentan ventajas sobre los CFGs. Los PCFGs fueron investigados por Booth (1969) y Salomaa (1969). Jelinek (1969) presenta el algoritmo de decodificación de pila, una variación de la búsqueda de Viterbi que puede ser usada para encontrar el análisis más probable con un PCFG. Baker (1979) introdujo el algoritmo de entrada-salida, y Lari y Young (1990) describieron sus usos y limitaciones. Charniak (1996) y Klein y Manning (2001) discutieron el análisis con las gramáticas **treebank**. Stolcke y Omohundro (1994) muestran cómo aprender las reglas de gramática con un modelo de Combinación Bayesiano. Otros algoritmos para PCFGs fueron presentados por Charniak (1993) y por Manning y Schütze (1999). Collins (1999) realiza un examen de la situación y da una explicación de uno de los programas más exitosos para el análisis estadístico.

Desgraciadamente, los PCFGs se comportan peor que los modelos simples de *n*-gram para una variedad de tareas porque los PCFGs no pueden representar la información asociada con palabras individuales. Para corregir esta deficiencia varios autores (Collins, 1996; Charniak, 1997; Hwa, 1998) han introducido versiones de **gramáticas probabilísticas lexicalizadas**, que combinan estadísticas de contexto libre y basadas en palabras.

GRAMÁTICAS PROBABILÍSTICAS LEXICALIZADAS

El Corpus Brown (Francis y Kucera, 1967) fue el primer esfuerzo de recoger un corpus equilibrado de texto de lenguaje empírico. Contenía alrededor de un millón de palabras, etiquetadas con parte del discurso. Originalmente se almacenó en 100.000 tarjetas perforadas. El treebank Penn (Marcus *et al*., 1993) es una colección de aproximadamente 1,6 millones de palabras, perforadas a mano en árboles. Se almacena en un CD. El Corpus Nacional Británico (Leech *et al*., 2001) se extiende a unos 100 millones de palabras. El World Wide Web tiene alrededor de un trillón de palabras y se almacena en alrededor de 10 millones de servidores.

El campo de **recuperación de información** está experimentando un nuevo crecimiento en interés, marcado por el amplio uso de la búsqueda en Internet. Robertson (1977) da una primera aproximación, e introduce el principio de ranking de probabilidad. Manning y Schütze (1999) dan una pequeña introducción al IR en el contexto de las aproximaciones estadísticas a NLP. Baeza-Yates y Ribeiro-Neto (1999) dan una descripción de propósito general, sustituyendo anteriores clásicos por Salton y McGill (1983) y por Frakes y Baeza-Yates (1992). El libro *Managing Gigabytes* (Witten *et al*., 1999) hace sólo lo que

el título dice: explica cómo indexar eficientemente, comprimir y hacer consultas sobre un corpora de un rango de gigabytes. La conferencia TREC, organizada por el National Institute of Standards and Technology (NIST), del gobierno de Estados Unidos, recoge una competición anual de sistemas IR y publica actas con los resultados. En los primeros siete años de concurso el rendimiento se ha doblado en términos generales.

El modelo más popular de IR es el **modelo de espacio vectorial** (Salton *et al.*, 1975). El trabajo de Salton dominó los primeros años de este campo. Hay dos alternativas de modelos probabilísticos. El primero que presentamos está basado en el trabajo de Ponte y Croft (1998). Modela la distribución de probabilidad conjunta $P(D, Q)$ en términos de $P(Q \mid D)$. Un modelo alternativo (Maron y Kuhns, 1960; Robertson y Sparck Jones, 1976) usa $P(D \mid Q)$. Lafferty y Zhai (2001) muestran que los modelos están basados en la misma distribución de probabilidad conjunta, pero que la elección del modelo tiene implicaciones en la preparación de parámetros. Nuestra presentación se deriva de la suya. Turtle y Croft (1992) comparan los diferentes modelos IR.

Brin y Page (1998) describen la implementación de un motor de búsqueda para el World Wide Web, incluyendo el algoritmo PAGERANK, una medida de la calidad del documento independiente de la consulta basada en un análisis de enlaces Web. Kleinberg (1999) describe cómo encontrar fuentes autorizadas en la Web usando análisis de enlaces. Silverstein *et al.* (1998) investigan un conjunto de billones de búsquedas en la Web. Kukich (1992) analiza la literatura de la corrección de la escritura. Porter (1980) describe el algoritmo clásico de procedencia basado en reglas, y Krovetz (1993) describe una versión basada en diccionario.

Manning y Schütze (1999) proporcionan una buena aproximación de la clasificación y agrupamiento de documentos. Joachims (2001) usa la teoría del aprendizaje estadístico y permite a las máquinas vectoriales dar un análisis teórico de cuándo una clasificación tendrá éxito. Apté *et al.* (1994) dan una precisión del 96 por ciento al clasificar los artículos de noticias de Reuters en la categoría de «Ganancias». Koller y Sahami (1997) dan una precisión de hasta el 95 por ciento usando un clasificador sencillo de Bayes, y hasta 98,6 por ciento usando otro clasificador de Bayes que tiene en cuenta algunas dependencias entre sus características. Lewis (1998) hace una revisión de 40 años de la aplicación de las técnicas sencillas de Bayes para la clasificación y recuperación del texto.

La revista *Information Retrieval* y las actas de la conferencia anual SIGIR recogen la mayoría de los desarrollos recientes en este campo.

Los programas de extracción de información más antiguos incluyen a GUS (Bobrow *et al.,* 1977) y a FRUMP (DeJong, 1982). Algunos de los diseños de los sistemas de extracción de información modernos están basados en las gramáticas de semántica de los años 70 y 80. Por ejemplo, un interfaz de un sistema de reserva de aerolíneas con una gramática semántica tendría categorías como *Lugar* y *Volar Hacia* en vez de *FN* y *FV*. *Véase* Birnbaum y Selfridge (1981) para una implementación de un sistema de este tipo basado en gramáticas semánticas.

La extracción de información reciente ha sido impulsada por la Conferencia anual sobre Comprensión del Mensaje (MUC), patrocinada por el gobierno de Estados Unidos. El sistema FASTUS fue hecho por Hobbs *et al.* (1997); la colección de artículos en los que aparece (Roche y Schabes, 1997) enumera otros sistemas que usan modelos de estado finitos.

En los años 30 Petr Troyanskii solicitó una patente para un «traductor automático» pero no había computadores disponibles para implementar sus ideas. En marzo de 1947, la fundación Warren Weaver Rockefeller escribió a Norbert Weiner, sugiriendo que la traducción automática podría ser posible. Continuando con el trabajo en criptografía y teoría de la información, Weaver escribió, «cuando miro un artículo en ruso, digo: "Esto realmente está escrito en inglés, pero lo han codificado con símbolos extraños. Ahora procederé a decodificarlo"». En la década siguiente, la comunidad intentó decodificarlo de esta manera. IBM exhibió un sistema rudimentario en 1954. Bar-Hillel (1960) y Locke y Booth (1955) describieron el entusiasmo de este período. La desilusión posterior con la traducción automática la describe Lindsay (1963), que también señala algunos de los obstáculos para la traducción automática relacionados con la interacción entre la sintaxis y la semántica y con la necesidad de conocimiento del mundo. El gobierno de Estados Unidos se desanimó por la falta de progresos, y un informe (ALPAC, 1966) concluyó que «no hay ningún estudio inmediato o predecible de traducción automática útil». Sin embargo, un cierto trabajo continuó, y el sistema SYSTRAN fue desarrollado por las Fuerzas Aéreas de Estados Unidos en 1970 y por la Comunidad Europea en 1976. El sistema de traducción del tiempo de TAUM-METEO también fue desarrollado en 1976 (Quinlan y O'Brien, 1992). A principios de los 80, la potencia de los computadores se había incrementado hasta el punto de que los informes ALPAC ya no fueron correctos nunca más. Voorhees (1993) habla sobre algunas aplicaciones recientes de traducción basadas en Wordnet. Un libro básico de introducción viene dado por Hutchins y Somers (1992).

La traducción automática estadística se remonta a la nota de Weaver de 1947, pero sólo al llegar a los años 80 se llevó a la práctica. Nuestra presentación está basada en el trabajo de Brown y sus colegas en IBM (Brown *et al.*, 1988, 1993). Es muy matemático, por lo que el tutorial que acompaña de Kevin Knight (1999) es un soplo de aire fresco. Un trabajo más reciente sobre la traducción automática estadística va más allá el modelo bigram y pasa por modelos que incluyen alguna sintaxis (Yamada y Knight, 2001). El trabajo más antiguo sobre segmentación de frases fue hecho por Palmer y Hearst (1994). Michel y Plamondon (1996) abarcan el alineamiento bilingüe de frases.

Hay dos libros excelentes sobre tratamiento probabilístico del lenguaje: Charniak (1993) es breve y puntual mientras que Manning y Schütze (1999) es comprensivo y actualizado. El trabajo sobre tratamiento práctico del lenguaje se presenta en la conferencia bianual Applied Natural Language Processing (ANLP), en la conferencia Empirical Methods in Natural Language Processing (EMNLP) y en la revista *Natural Language Engineering*. SIGIR patrocina un boletín de noticias y una conferencia anual sobre recuperación de información.

EJERCICIOS

23.1 (Adaptado de Jurafsky y Martin (2000)). En este ejercicio desarrollaremos un clasificador para autenticidad: dado un texto, trataremos de determinar quién de entre dos candidatos autores escribió el texto. Obtener muestras de texto de dos autores diferentes. Separarlos en conjuntos de preparación y de prueba. Ahora preparar un modelo de palabra unigram para cada autor del conjunto de preparación. Finalmente, para cada con-

junto de prueba, calcular su probabilidad de acuerdo con cada modelo unigram y asignarla al modelo más probable. Asegurarse de la precisión de esta técnica. ¿Puede mejorar su precisión con características adicionales? Este subcampo de la lingüística se llama **estilometría**; sus éxitos incluyen la identificación del autor de *Federalist Papers* (Mosteller y Wallace, 1964) y algunos trabajos litigados a Shakespeare (Foster, 1989).

ESTILOMETRÍA

23.2 Este ejercicio explora la calidad de un modelo *n*-gram de lenguaje. Buscar o crear un corpus monolingüe de alrededor de 100.000 palabras. Segmentarlo en palabras y calcular la frecuencia de cada palabra. ¿Cuántas palabras distintas hay? Dibujar la frecuencia de las palabras frente a su rango (primera, segunda, tercera...) en una escala log-logarítmica. Además, contar las frecuencias de bigrams (dos palabras consecutivas) y trigrams (tres palabras consecutivas). Ahora usar esas frecuencias para generar el lenguaje: a partir de los modelos unigram, bigram y trigram generar un texto de 100 palabras utilizando elecciones aleatorias de acuerdo con las frecuencias calculadas. Comparar los tres textos generados con el lenguaje actual. Finalmente, calcular la perplejidad de cada modelo.

23.3 Este ejercicio afecta a la detección de un email de tipo spam. El spam se define como mensajes por email comerciales no solicitados que se envían en masa. Trabajar con el spam es un problema muy pesado para muchos usuarios de email, por lo que una manera segura de eliminarlos sería una alegría. Crear un corpus de email spam y otro corpus de email no spam. Examinar cada corpus y decidir qué características parecen ser útiles para su clasificación: ¿palabras unigram?, ¿bigram? ¿longitud del mensaje, remitente, tiempo que tarda en llegar? Entonces preparar un algoritmo de clasificación con (árboles de decisión, Bayes sencillo, y otros algoritmos de su elección) sobre un conjunto preparado y dar un informe de su precisión para un conjunto de pruebas.

23.4 Crear un conjunto de prueba de cinco peticiones, y poner en práctica estas peticiones en los tres mayores motores de búsqueda de la Web. Evaluar cada una para una precisión de uno, tres y 10 documentos devueltos y para un ranking medio recíproco. Intentar explicar las diferencias.

23.5 Tratar de asegurar cuáles de los motores de búsqueda del ejercicio anterior utilizan categorías, procedencia, sinónimos y corrección de escritura.

23.6 Estimar cuánto espacio de almacenamiento es necesario para el índice de un corpus de un billón de páginas Web. Muestre las suposiciones que ha hecho.

23.7 Escriba una expresión regular o un programa corto para extraer nombres de empresas. Pruébelo en un corpus de artículos de noticias sobre negocios. Informe sobre el resultado y su precisión.

23.8 Seleccione cinco frases y envíelas a un servicio de traducción online. Tradúzcalas del inglés a otro idioma y otra vez al inglés. Clasifique las frases resultantes por gramaticalidad y preservación del significado. Repita el proceso; ¿los resultados de la segunda ronda de iteraciones son peores resultados o son los mismos? ¿La elección del lenguaje intermedio puede producir diferencias en la calidad de los resultados?

23.9 Recoja algunos ejemplos de expresiones temporales, tales como «las dos en punto», «media noche» y «12:46». Piense también algunos ejemplos que no sean gramati-

cales, tales como «las trece en punto» o «las dos y cuarto y media». Escriba una gramática para el lenguaje temporal.

23.10 (Adaptado de Knight (1999)). El modelo de traducción automática «IBM Model 3» asume que, después de que un modelo de elección de palabra proponga una lista de palabras y el modelo de compensación proponga posibles permutaciones de las palabras, el modelo de lenguaje puede elegir la mejor permutación. Este ejercicio investiga la sensibilidad es esta suposición. Trate de ordenar las siguientes frases propuestas en el orden correcto:

- haber programando un visto nunca yo idioma mejor
- ama john mary
- es la comunicación intercambio de intencional información promovida por sobre la producción percepción de y signos a partir de dibujo un de sistema signos convencional compartido.

¿Cuáles se podrían hacer? ¿Qué tipo de conocimiento ha utilizado? Prepare un modelo bigram a partir del corpus de preparación, y utilícelo para encontrar la permutación de mayor probabilidad de algunas frases para el corpus de prueba. Informe sobre la precisión de este modelo.

23.11 Si mira en un diccionario de inglés-francés, la traducción de «hear» es el verbo «entendre». Pero si prepara el modelo «IBM Model 3» del Hansard canadiense, la traducción más probable para «hear» es «Bravo». Explique por qué ocurre esto y estime cuál es la distribución de fertilidad que podría tener «hear». (*Ayuda:* podría querer mirar algún texto Hansard. Intente con una búsqueda Web de [Hansard hear].)

24 Percepción

Donde conectamos el computador con la cruda realidad.

PERCEPCIÓN

SENSORES

La **percepción** proporciona a los agentes información sobre el mundo en el que habitan. El proceso de percepción se origina en los **sensores**. Un sensor es un dispositivo que capta algún aspecto del entorno y lo pasa como entrada al programa agente. El sensor puede ser tan simple como un sensor de un bit que detecta si un interruptor está encendido o apagado o tan complejo como la retina del ojo humano, que contiene más de 100 millones de elementos fotosensibles. En este capítulo, nos centraremos en la visión, ya que es con mucha diferencia el sentido más útil para desenvolverse en un mundo físico.

24.1 Introducción

Los agentes artificiales disponen de una gran variedad de modalidades de percepción. Las que comparten con los humanos son la visión, el oído y el tacto. El oído, al menos en lo que respecta a la comprensión del habla, fue tratado en el Apartado 15.6. El **sentido del tacto** se discute en el Capítulo 25, donde examinamos su uso en la habilidad de manipulación de objetos por robots. El resto de este capítulo tratará la visión. Los robots pueden tratar formas de percepción de las que no disponemos los humanos, al menos sin ayuda, como son las ondas de radio, los infrarrojos, el GPS y las señales inalámbricas. Algunos robots realizan una **percepción activa**, lo que significa que en-

vían una señal, de radar o de ultrasonido por ejemplo, y captan la reflexión de esta señal en el entorno.

Un agente puede utilizar sus percepciones de dos formas. En el enfoque basado en la **extracción de características**, los agentes detectan un pequeño número de características en su entrada sensorial y las pasan directamente a su programa agente, que puede responder de forma reactiva a ellas, o puede combinarlas con otra información. El agente *wumpus* trabajaba de este modo, con cinco sensores, cada uno de los cuales extraía una característica de un bit. Se sabe que una mosca extrae características del flujo óptico y las envía directamente a los músculos que la ayudan a orientarse, permitiéndole reaccionar y cambiar la dirección en 30 milisegundos.

La alternativa es un enfoque **basado en un modelo**, en el cual el estímulo sensorial se utiliza para reconstruir un modelo del mundo. En esta aproximación comenzamos con una función f que realiza la correspondencia entre el estado del mundo, W, y el estímulo, S, que el mundo producirá:

$$S = f(W)$$

GRÁFICOS POR COMPUTADOR

La función f se define en términos físicos y ópticos, y se ha llegado a comprender bastante bien. Generar S a partir de f y un mundo real o imaginario W es un problema propio de los **gráficos por computador**. La visión por computador es de alguna manera la inversa de los gráficos por computador: dados f y S, tratamos de computar W mediante

$$W = f^{-1}(S)$$

Desafortunadamente, no es posible obtener una inversa adecuada para f. Por un lado, no es posible ver detrás de los objetos, por lo que no podemos reconstruir todos los aspectos del mundo a partir del estímulo. Además, incluso la parte que *podemos* ver es tremendamente ambigua: sin información adicional no es posible determinar si S es una imagen de un muñeco de Godzilla destruyendo la maqueta de un edificio de unos pocos centímetros de altura, o un monstruo real destruyendo un edificio de decenas de metros. Podemos tratar algunas de estas cuestiones construyendo una distribución de probabilidad sobre los mundos, en lugar de tratar de encontrar un único mundo:

$$P(W) = P(W|S)P(S)$$

Un inconveniente aún mayor de este tipo de modelos es que tratan de resolver el problema de una forma excesivamente complicada. Pensemos que en los gráficos por computador la generación de un único fotograma de una película puede suponer varias horas de cálculos, que para cada segundo son necesarios 24 fotogramas, y que la obtención de f^{-1} es mucho más compleja que la de f. Estos cálculos son claramente excesivos para un supercomputador, y no digamos para una simple mosca que debe reaccionar en tiempo real. Afortunadamente, el agente no necesita un modelo con el nivel de detalle usado en los gráficos por computador fotorrealistas. El agente sólo necesita conocer si hay un tigre escondido entre la maleza, y no la localización y orientación exactas de cada pelo de su lomo.

En la mayor parte de este capítulo, trataremos de cómo reconocer objetos, tales como tigres, y veremos distintas formas de hacerlo sin representar cada uno de sus detalles. En el Apartado 24.2 estudiamos el proceso de formación de la imagen, definiendo al-

gunos aspectos de la función $f(W)$. En primer lugar nos fijaremos en la geometría del proceso. Veremos que la luz se refleja en los objetos situados en la escena, e incide sobre los puntos del plano de imagen del sensor del agente. La geometría explica por qué un objeto de gran tamaño visto desde lejos es igual que otro pequeño con la misma forma visto a poca distancia. Después nos fijaremos en la fotometría del proceso, que describe cómo la luz de la escena determina el brillo de los puntos en la imagen. Juntos, geometría y fotometría nos darán un modelo de cómo los objetos del mundo se plasman en una matriz bidimensional de píxeles.

Una vez comprendido cómo se forman las imágenes, pasaremos a ver cómo se procesan. El flujo de información en el proceso visual tanto en humanos como en computadores puede dividirse en tres fases. En la visión primitiva o de bajo nivel (Apartado 24.3) se suaviza la imagen original y se elimina el ruido, y se extraen las características de la imagen bidimensional, en particular las aristas existentes entre regiones de la imagen. En la visión de nivel medio se agrupan estas aristas para formar regiones bidimensionales. En la visión de alto nivel (Apartado 24.4), las regiones bidimensionales son reconocidas como objetos reales del mundo (Apartado 24.5). Estudiaremos varios elementos clave en la imagen, tales como movimiento, visión estereoscópica, texturas, sombreados y contornos, y los aprovecharemos para este fin. El reconocimiento de objetos es importante en la naturaleza para que un agente pueda detectar tigres, y es importante también para que los robots industriales puedan distinguir las tuercas de los tornillos. Por último, el Apartado 24.6 describe cómo el reconocimiento de objetos puede ayudarnos a llevar a cabo tareas útiles, tales como manipulación y navegación. La manipulación significa ser capaz de agarrar y usar herramientas y otros objetos. La navegación se refiere a la capacidad para moverse de un lugar a otro sin tropezar con nada. Teniendo en mente estas tareas podemos asegurar que un agente construye sólo aquella parte del modelo que necesita para alcanzar sus metas.

24.2 Formación de la imagen

ESCENA

IMAGEN

PÍXELES

La visión recoge la luz reflejada por los objetos en la **escena** y crea una **imagen** bidimensional sobre un plano de imagen. El plano de imagen está recubierto con material fotosensible: moléculas de rodopsina en la retina, haluro de plata en una película fotográfica o una matriz CCD (*Charge-Coupled Device*) en una cámara digital. Cada punto de un CCD integra los electrones liberados por la absorción de fotones durante un período de tiempo establecido. En una cámara digital el plano de imagen se subdivide en una rejilla rectangular de unos pocos millones de **píxeles**. El ojo tiene una matriz de píxeles similar que consiste en unos 100 millones de bastoncillos y cinco millones de conos, dispuestos en una malla hexagonal.

La escena real es muy grande comparada con el pequeño plano de imagen, por lo que es necesario disponer de alguna forma de enfocar la luz sobre este plano. Esta labor se puede realizar con lentes o sin ellas. En cualquier caso, la clave está en definir la geometría de manera que podamos decir dónde se encontrará cada punto de la escena sobre el plano de imagen.

Imágenes sin lentes: la cámara de orificio o pinhole

CÁMARA DE ORIFICIO

La manera más sencilla de formar una imagen es con una **cámara de orificio** o pinhole, que consiste en un orificio O, sobre el frontal de una caja, y un plano de imagen en la parte trasera de la misma (Figura 24.1). Utilizaremos un sistema de coordenadas tridimensional con el origen en O y consideraremos un punto P de la escena con coordenadas (X, Y, Z). La proyección de P sobre el plano de imagen es el punto P' de coordenadas (x, y, z). Si f es la distancia del orificio al plano de imagen, por triángulos semejantes podemos derivar las siguientes ecuaciones:

$$\frac{-x}{f} = \frac{X}{Z}, \frac{-y}{f} = \frac{Y}{Z} \quad \Rightarrow \quad x = \frac{-fX}{Z}, y = \frac{-fY}{Z}$$

PROYECCIÓN PERSPECTIVA

Estas ecuaciones definen el proceso de formación de una imagen, conocido como **proyección perspectiva**. Obsérvese que la Z del denominador significa que cuanto más lejano se encuentra el objeto, menor es su imagen. Observemos también que el signo negativo significa que la imagen está *invertida*, tanto de izquierda a derecha como de arriba abajo, en comparación con la escena.

Bajo una proyección perspectiva, las líneas paralelas convergen en un punto en el horizonte (pensemos en los raíles de un tren). Veamos por qué ocurre esto. Una línea en la escena que pase por el punto (X_0, Y_0, Z_0) en la dirección (U, V, W) puede describirse como el conjunto de puntos $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda \mathrm{W})$, con λ tomando valores entre $-\infty$ y $+\infty$. La proyección de un punto $P\lambda$ de esta recta sobre el plano de imagen viene dada por

$$\left(f\frac{X_0 + \lambda U}{Z_0 + \lambda \mathrm{W}}, f\frac{Y_0 + \lambda V}{Z_0 + \lambda \mathrm{W}} \right)$$

PUNTO DE FUGA

Conforme $\lambda \to +\infty$ o $\lambda \to -\infty$, este punto se convierte en $p_\infty = (fU/W, fV/W)$ si $W \neq 0$. Llamamos a p_∞ **punto de fuga** asociado a la familia de rectas con dirección (U, V, W). Las líneas con la misma dirección comparten el mismo punto de fuga.

Figura 24.1 Geometría de la formación de imágenes en una cámara de orificio.

PROYECCIÓN ORTOGRÁFICA

Si el objeto tiene poca profundidad relativa, comparado con la distancia a la cámara, podemos aproximar la proyección perspectiva mediante una **proyección ortográfica a escala**. La idea es la siguiente: si la profundidad Z de los puntos del objeto varía dentro de un rango $Z_0 \pm \Delta Z$, con $\Delta Z \ll Z_0$, el factor de escala en perspectiva f/Z puede aproximarse mediante una constante $s = f/Z_0$. Las ecuaciones para la proyección de las coordenadas de la escena (X, Y, Z) al plano de imagen se convierten en $x = sX$ e $y = sY$. Obsérvese que la proyección ortográfica a escala es una aproximación que es válida sólo para aquellas partes de la escena con poca variación de profundidad interna; no debe utilizarse para estudiar propiedades «globales». Un ejemplo para convencerse de la necesidad de precaución: bajo una proyección ortográfica, las líneas paralelas permanecen paralelas en vez de converger en el punto de fuga.

Sistemas de lentes

LENTES

Los ojos de los vertebrados y las cámaras modernas utilizan **lentes**. Una lente es mucho mayor que un orificio y permite la entrada de más luz. Esto provoca que no toda la escena pueda estar perfectamente enfocada a la vez. La imagen de un objeto que se encuentra a distancia Z en la escena se produce a una distancia fija Z' de la lente, donde la relación entre Z y Z' viene dada por la ecuación de la lente

$$\frac{1}{Z} + \frac{1}{Z'} = \frac{1}{f}$$

en la que f es la longitud focal de la lente. Dada una cierta distancia Z_0' elegida entre el punto nodal de la lente y el plano de imagen, los puntos de la escena con profundidad alrededor de Z_0, donde Z_0 es la distancia al objeto, se visualizan razonablemente bien enfocados. A este rango de profundidades en una escena se le conoce como **profundidad de campo**.

PROFUNDIDAD DE CAMPO

Obsérvese que, dado que la distancia Z al objeto es habitualmente mucho mayor que la distancia a la imagen Z' o f, a menudo se hace la siguiente aproximación:

$$\frac{1}{Z} + \frac{1}{Z'} \approx \frac{1}{Z'} \Rightarrow \frac{1}{Z'} \approx \frac{1}{f}$$

De este modo la distancia a la imagen $Z' \approx f$. Por lo tanto podemos continuar utilizando las ecuaciones de proyección perspectiva de la cámara de orificio para describir la geometría de la formación de imágenes en un sistema con lentes.

Para enfocar objetos que están a diferentes distancias Z, la lente de un ojo (*véase* Figura 24.2) cambia de forma, mientras que la lente de una cámara se mueve en dirección Z.

Luz: la fotometría de la formación de imágenes

FOTOMETRÍA

La luz es un requisito imprescindible para la visión: sin luz, todas las imágenes estarían totalmente oscuras sin importar el interés de la escena. La **fotometría** es el estudio de la luz. Para nuestros propósitos, consistirá en modelar cómo la luz de la escena se co-

Figura 24.2 Sección transversal horizontal del ojo humano.

rresponde con la intensidad de luz en el plano de imagen a lo largo del tiempo, lo que denotaremos como $I(x, y)$[1]. Un sistema de visión aplica este modelo en sentido inverso, es decir, partiendo de la intensidad que la imagen pretende obtener de las propiedades del mundo real. La Figura 24.3 muestra una imagen digitalizada de una grapadora sobre un escritorio, y un primer plano de un bloque de 12×12 píxeles extraído de la imagen de la grapadora. Un programa de computador que intente interpretar la imagen debería empezar con una matriz de valores de intensidad como esta.

El brillo de un píxel en la imagen es proporcional a la cantidad de luz que llega a la cámara proveniente de la porción de superficie de la escena que se proyecta en ese pí-

195	209	221	235	249	251	254	255	250	241	247	248
210	236	249	254	255	254	225	226	212	204	236	211
164	172	180	192	241	251	255	255	255	255	235	190
167	164	171	170	179	189	208	244	254	255	251	234
162	167	166	169	169	170	176	185	196	232	249	254
153	157	160	162	169	170	168	169	171	176	185	218
126	135	143	147	156	157	160	166	167	171	168	170
103	107	118	125	133	145	151	156	158	159	163	164
095	095	097	101	115	124	132	142	117	122	124	161
093	093	093	093	095	099	105	118	125	135	143	119
093	093	093	093	093	093	095	097	101	109	119	132
095	093	093	093	093	093	093	093	093	093	093	119

(a) (b) (c)

Figura 24.3 (a) Una fotografía de una grapadora sobre un escritorio. (b) Vista aumentada de un bloque de 12×12 píxeles de la imagen (a). (c) Los valores de intensidad de imagen asociados en una escala del 0 al 255.

[1] Cuando nos interesen los cambios a lo largo del tiempo utilizaremos $I(x, y, t)$.

xel. Esta luz depende a su vez de las propiedades reflectantes de la porción de superficie y de la posición y distribución de las fuentes de luz en la escena. También depende en parte de las propiedades reflectantes del resto de la escena, ya que otras superficies pueden servir como fuentes de luz indirectas al reflejar la luz.

REFLEXIÓN ESPECULAR

REFLEXIÓN DIFUSA

Podemos modelar dos tipos diferentes de reflexión. La **reflexión especular** hace referencia a la luz que es reflejada por la superficie exterior del objeto, y cumple la restricción de que el ángulo de reflexión de la luz es igual a su ángulo de incidencia. Este es el comportamiento de un espejo perfecto. La **reflexión difusa** se produce cuando la luz penetra la superficie del objeto, es absorbida por él, y luego es reemitida. Para una superficie perfectamente difusora (o **Lambertiana**) la luz es emitida con igual intensidad en todas las direcciones. La intensidad depende sólo del ángulo de incidencia de la fuente de luz: una fuente de luz situada directamente sobre el objeto hará que éste la refleje en su mayoría, mientras que una fuente de luz que es casi paralela al objeto hará que casi no se refleje luz. Entre estos dos extremos la intensidad de luz reflejada, I, obedece la ley del coseno de Lambert

$$I = kI_0 \cos\theta$$

donde I_0 es la intensidad de la fuente de luz, θ es el ángulo entre la fuente de luz y la normal a la superficie, y k es una constante denominada **albedo**[2], que depende de las propiedades reflectantes de la superficie. Varía entre 0 (para superficies de color negro perfecto) a 1 (para superficies de color blanco puro).

En la vida real, las superficies presentan una combinación de propiedades difusas y especulares. El modelado de esta combinación es la esencia de los gráficos por computador. La visualización de imágenes realistas se lleva a cabo habitualmente mediante el trazado de rayos, que trata de simular el proceso físico que realiza la luz que se origina en las fuentes de luz y se refleja una y otra vez sobre los objetos de la escena.

Color: la espectrofotometría de la formación de imágenes

En la Figura 24.3 mostrábamos una imagen en blanco y negro que obvia el hecho de que la luz visible nos llega en un amplio rango de longitudes de onda (varían desde 400 nm en el extremo violeta del espectro de luz hasta 700 nm en el extremo rojo). Algunas luces están formadas por una única longitud de onda pura y se corresponden con uno de los colores del arco iris. Pero las demás luces son una mezcla de longitudes de onda. ¿Significa esto que necesitamos una mezcla de valores para un nivel de intensidad $I(x, y)$ en vez de un único valor? Si quisiéramos representar la física de la luz de forma exacta, en efecto lo necesitaríamos. Pero si sólo queremos reproducir la percepción que los humanos (y otros muchos vertebrados) tenemos de la luz, podemos llegar a un compromiso. Algunos experimentos (si nos remontamos a los trabajos de Thomas Young en 1801) han demostrado que cualquier mezcla de longitudes de onda, sin importar su complejidad, puede reproducirse con la mezcla de únicamente tres colores primarios. Es decir, si te-

[2] *Nota del traductor:* Razón entre la energía luminosa que difunde por reflexión una superficie y la energía incidente.

nemos un generador de luz que puede combinar de forma lineal tres longitudes de onda (de forma habitual, se eligen el rojo (700 nm), el verde (546 nm) y el azul (436 nm)) entonces ajustando los valores para incrementar o decrementar la cantidad de cada color podemos obtener cualquier combinación de longitudes de onda, desde el punto de vista de la percepción humana. Este hecho experimental supone que las imágenes pueden representarse como un vector con únicamente tres valores de intensidad por píxel: uno para cada una de las tres longitudes de onda primarias. En la práctica, un byte para cada uno de estos valores resulta suficiente para una reproducción de la imagen con una fidelidad muy alta. Esta percepción tricromática del color está relacionada con el hecho de que la retina tiene tres tipos de conos con picos máximos de receptividad en 650 nm, 530 nm y 430 nm respectivamente, aunque los detalles exactos del proceso son más complejos que una simple relación uno a uno.

24.3 Operaciones de procesamiento de imagen a bajo nivel

Hemos visto cómo la luz se refleja en los objetos de la escena para formar una imagen consistente en, por ejemplo, cinco millones de píxeles de tres bytes. Como ocurre con cualquier sensor la imagen tendrá ruido y, en cualquier caso, contendrá mucha información que tratar. En esta sección veremos qué hacer con los datos de la imagen para que sea más fácil tratar con ellos. En primer lugar veremos las operaciones de suavizado de la imagen para reducir el ruido, y las de detección de aristas en la imagen. Estas se conocen como operaciones de bajo nivel porque son las primeras en una secuencia de operaciones. Las operaciones de visión tempranas se caracterizan por su naturaleza local (pueden llevarse a cabo en una parte de la imagen sin preocuparse de lo que hay más allá de unos cuantos píxeles) y por no necesitar la compresión de la escena: podemos suavizar una imagen y detectar sus aristas sin tener ni idea de los objetos que hay en la escena. Esto hace que las operaciones de bajo nivel sean candidatas perfectas para su implementación sobre hardware paralelo, tanto directamente sobre el chip como mediante una simulación. A continuación veremos una operación de nivel medio, la segmentación de imágenes en regiones. Esta fase del proceso todavía opera sobre la imagen, no sobre la escena, pero incluye procesamiento no local.

En el Apartado 15.2, **suavizar** significaba predecir el valor de una variable de estado en algún momento t del pasado, dados sus valores en otros momentos desde t hasta el presente. Ahora aplicamos la misma idea al dominio espacial en vez de al temporal: suavizar significa predecir el valor de un píxel dados los píxeles de alrededor. Obsérvese que debemos tener clara la diferencia entre el valor observado medido en el píxel, y el valor real que deberíamos haber medido en ese píxel. Este valor puede ser distinto debido a errores de medida aleatorios o debido a un fallo del sistema: el receptor en el CCD puede haber fallado.

Una forma de suavizar una imagen es asignar a cada píxel la media de sus vecinos. Este procedimiento tenderá a eliminar los valores extremos. Pero ¿cuántos vecinos de-

FILTRO GAUSSIANO

bemos considerar, uno o dos o más? Una solución posible que funciona bien para eliminar el ruido Gaussiano es una media ponderada utilizando el **filtro Gaussiano**. Recordemos que la función de Gauss con desviación tipica σ es:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-x^2/2\sigma^2} \qquad \text{en una dimensión, o}$$

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad \text{en dos dimensiones}$$

Aplicar un filtro Gaussiano significa reemplazar la intensidad $I(x_0, y_0)$ con la suma, para todos los píxeles (x, y), de $I(x, y)G_\sigma(d)$, donde d es la distancia de (x_0, y_0) a (x, y). Este tipo de suma ponderada es tan común que hay un nombre y una notación especiales para ella. Decimos que la función h es la **convolución** de dos funciones f y g (se denota como $h = f * g$) si tenemos

CONVOLUCIÓN

$$h(x) = \sum_{u=-\infty}^{+\infty} f(u)\, g(x-u) \qquad \text{en una dimensión, o}$$

$$h(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v)\, g(x-u, y-v) \qquad \text{en dos dimensiones}$$

Por lo tanto, la función de suavizado se realiza mediante una convolución de la imagen con la Gaussiana, $I * G_\sigma$. Una σ de un píxel es suficiente para suavizar pequeñas cantidades de ruido, mientras que dos píxeles suavizarán una cantidad mucho mayor, pero con pérdida de algunos detalles. Dado que la influencia de la Gaussiana se desvanece con la distancia, en la práctica podemos reemplazar el $\pm\infty$ de los sumatorios por valores alrededor de $\pm 3\sigma$.

Detección de aristas

ARISTAS

El siguiente paso en la visión de bajo nivel es la detección de aristas en el plano de imagen. Las **aristas** son líneas rectas o curvas en el plano de imagen que marcan un cambio «significativo» en el brillo de la imagen. El objetivo de la detección de aristas es abstraerse de la confusa imagen de varios megabytes y llegar a una representación más compacta y abstracta como la que se presenta en la Figura 24.4. La idea de partida es que los contornos formados por aristas de la imagen se corresponden con los contornos más importantes de la escena. En la figura tenemos tres ejemplos de discontinuidad en profundidad, etiquetados como 1; dos de discontinuidad en la orientación de las superficies, etiquetados como 2; uno de discontinuidad en la reflectancia, etiquetado como 3 y otro de discontinuidad en la iluminación (sombra), etiquetada como 4. La detección de aristas trata sólo con la imagen, y por lo tanto no distingue estos tipos diferentes de discontinuidades en la escena, aunque un procesamiento posterior sí lo hará.

La Figura 24.5(a) muestra una imagen de una escena que contiene una grapadora sobre un escritorio, y (b) muestra la salida de un algoritmo de detección de aristas para esta imagen. Como puede observarse, hay diferencias entre la salida y un dibujo con líneas ideal. Las aristas de los elementos más pequeños no se alinean todas con las demás, hay

Figura 24.4 Diferentes tipos de aristas: (1) discontinuidades en profundidad; (2) discontinuidades en la orientación de las superficies; (3) discontinuidades en reflectancia; (4) discontinuidades en iluminación (sombras).

Figura 24.5 (a) Fotografía de una grapadora. (b) Aristas obtenidas a partir de (a).

discontinuidades entre las que no aparecen aristas, y hay algunas aristas que introducen ruido al no corresponderse con nada significativo de la escena. Etapas posteriores del procesamiento tendrán que corregir estos errores. ¿Cómo detectamos aristas en una imagen? Consideremos el perfil de los valores de intensidad de una imagen a lo largo de una sección unidimensional que cruce perpendicularmente una arista, por ejemplo, la que se encuentra entre el borde izquierdo del escritorio y la pared. Se parecerá a algo como lo que se muestra en la Figura 24.6(a). La localización de la arista corresponde a $x = 50$.

Puesto que las aristas se corresponden con zonas de la imagen en las que el brillo sufre un cambio brusco, una idea simplista sería diferenciar la imagen y buscar aquellas zonas en las que la magnitud de la derivada $I'(x)$ es grande. Esto casi funciona.

Figura 24.6 Arriba: perfil de intensidades $I(x)$ a lo largo de una sección unidimensional que cruza una arista. Centro: derivada de la intensidad, $I'(x)$. Los valores altos de esta función corresponden a aristas, pero esta función es ruidosa. Abajo: derivada de una versión suavizada de la intensidad, $(I * G_\sigma)'$, que puede calcularse en un único paso como la convolución $I * G'_\sigma$. La arista candidata que aparecía en $x = 75$ ha desaparecido.

En la Figura 24.6(b), vemos que, aunque hay un pico en $x = 50$, hay también picos secundarios en otros lugares (por ejemplo, en $x = 75$) que pueden confundirse con aristas reales. Esto se produce por la presencia de ruido en la imagen. Si suavizamos la imagen en primer lugar, los picos falsos se reducen, como podemos ver en (c).

Podemos aprovechar para realizar una optimización aquí: es posible combinar el suavizado y la detección de aristas en una única operación. Existe un teorema que nos dice que para cualquier par de funciones f y g, la derivada de la convolución, $(f * g)'$, es igual a la convolución de f con la derivada de g, $f * (g)'$. Por ello, en vez de suavizar la imagen y diferenciar después, podemos calcular la convolución de la imagen con la derivada de la función Gaussiana de suavizado, G'_σ. Así, en una dimensión el algoritmo de búsqueda de aristas es:

1. Calcular la convolución de la imagen I con G'_σ para obtener R.
2. Marcar como aristas aquellos picos de $\|R(x)\|$ que superan un umbral T predeterminado. El umbral se elige de tal manera que se eliminen los picos falsos debidos al ruido.

En dos dimensiones las aristas pueden encontrarse orientadas con cualquier ángulo θ. Para detectar aristas verticales, optaremos por una estrategia obvia: calcular la convo-

lución con $G'_\sigma(x)G_\sigma(y)$. En la dirección y, el efecto es simplemente un suavizado (debido a la convolución Gaussiana), y en la dirección x, el efecto es el de una diferenciación acompañada con un suavizado. El algoritmo para detectar aristas verticales queda de la siguiente manera:

1. Calcular la convolución de la imagen $I(x, y)$ con $f_V(x, y) = G'_\sigma(x)G_\sigma(y)$ para obtener $R_V(x, y)$.
2. Marcar aquellos picos de $\|R_V(x,y)\|$ que superan un umbral T predeterminado.

Para detectar una arista con una orientación arbitraria, necesitamos calcular la convolución de la imagen con dos filtros: $f_V = G'_\sigma(x)G_\sigma(y)$ y $f_H = G'_\sigma(y)G_\sigma(x)$, que es justamente f_V rotado 90°. El algoritmo para detectar aristas con orientaciones arbitrarias queda de la siguiente manera:

1. Calcular la convolución de la imagen $I(x,y)$ con $f_V(x,y)$ y $f_H(x,y)$ para obtener $R_V(x,y)$ y $R_H(x,y)$, respectivamente. Definimos $R(x,y)= R_V^2(x, y) + R_H^2(x, y)$.
2. Marcar aquellos picos de $\|R(x, y)\|$ que superan un umbral T predeterminado.

Una vez que hemos marcado los píxeles de las aristas utilizando este algoritmo, la siguiente etapa es unir aquellos píxeles que pertenecen a las mismas aristas. Se puede realizar asumiendo que dos píxeles vecinos cualesquiera que pertenezcan a las aristas y tengan orientaciones consistentes deben pertenecer a la misma arista. Este proceso se conoce como **detección de aristas de Canny**, debido a su inventor John Canny.

DETECCIÓN DE ARISTAS DE CANNY

Una vez detectadas, las aristas forman la base de muchos procesos posteriores: podemos utilizarlas para realizar procesos estereoscópicos, detectar movimiento, o reconocer objetos.

Segmentación de la imagen

Los humanos *organizamos* la entrada perceptual: en lugar de una colección de valores de brillo asociados a fotorreceptores individuales, percibimos un número de grupos visuales, asociados habitualmente con objetos o partes de objetos. Esta habilidad es igualmente importante en la visión por computador.

SEGMENTACIÓN

La **segmentación** es el proceso de dividir la imagen en grupos, basado en las similitudes entre los píxeles. La idea básica es la siguiente: cada píxel de la imagen puede asociarse con ciertas propiedades visuales, tales como el brillo, el color y la textura[3]. Dentro de un objeto, o de una parte del objeto, estos atributos varían relativamente poco, mientras que al cruzar los bordes entre objetos hay habitualmente un gran cambio en unos u otros de estos atributos. Necesitamos encontrar una partición de la imagen en conjuntos de píxeles tal que estas restricciones se satisfagan lo mejor posible.

Hay varias maneras de formalizar este proceso intuitivo de forma matemática. Por ejemplo, Shi y Malik (2000) lo plantean como un problema de partición de un grafo. Los nodos del grafo se corresponden con los píxeles, y las aristas con conexiones entre pí-

[3] Las propiedades de textura se basan en estadísticos medidos en pequeñas áreas centradas en el píxel.

xeles. El peso W_{ij} de la arista que conecta un par de píxeles i y j depende de la similitud en brillo, color, textura, etc., existente entre los dos píxeles. Después se buscan las particiones que minimizan un criterio de *división normalizado*. A groso modo, el criterio para dividir el grafo es minimizar la suma de los pesos de las conexiones entre grupos y maximizar la suma de los pesos de las conexiones dentro de los grupos.

Una segmentación basada únicamente en atributos locales y de bajo nivel, tales como el brillo y el color, es un proceso expuesto a muchos errores. Para encontrar los bordes asociados a los objetos de una forma fiable, debemos incorporar también conocimiento de alto nivel referido a los tipos de objetos que esperamos encontrar en una escena. El formalismo del modelo de Markov Oculto se utiliza en este sentido para el reconocimiento del habla; en el contexto de las imágenes un marco de trabajo unificado de este tipo sigue siendo un campo de investigación activo. En cualquier caso, el conocimiento de los objetos a alto nivel es el tema de la siguiente sección.

24.4 Extracción de información tridimensional

En esta sección mostramos cómo pasar de una imagen bidimensional a una representación tridimensional de la escena. Es importante el razonamiento sobre la escena, ya que, después de todo, el agente vive en el mundo real, no en el plano de imagen, y el objetivo de la visión es ser capaz de interactuar con los objetos del mundo. Sin embargo, la mayoría de los agentes necesita sólo una representación abstracta limitada de ciertos aspectos de la escena, y no de cada detalle. Los algoritmos que hemos visto en el resto del libro para tratar con el mundo dependen de que tengamos una descripción concisa de los objetos, no enumeraciones exhaustivas de cada porción de la superficie tridimensional.

RECONOCIMIENTO DE OBJETOS

En primer lugar trataremos el **reconocimiento de objetos**, el proceso de convertir las características de la imagen (tales como las aristas) en modelos de objetos conocidos (tales como grapadoras). El reconocimiento de objetos se compone de tres pasos: La segmentación de la escena en objetos distintos, la determinación de la posición y la orientación de cada objeto relativas al observador, y la determinación de la forma de cada objeto.

POSTURA

Averiguar la posición y la orientación de un objeto en relación con el observador (la denominada **postura** del objeto) es de gran importancia para las tareas de manipulación y navegación. Para moverse a través del suelo de una fábrica repleta de objetos y trabajadores, se necesita conocer la localización de los obstáculos, de manera que podamos planificar una trayectoria que los evite. Si queremos recoger y agarrar un objeto, necesitamos conocer su posición relativa a la mano, de manera que podamos generar una trayectoria de movimientos apropiada. Las acciones de manipulación y navegación se realizan habitualmente a través de un bucle de control de la posición: la información sensorial proporciona una retroalimentación para modificar el movimiento del robot, o el movimiento del brazo del robot.

Especifiquemos la posición y la orientación en términos matemáticos. La posición de un punto P en una escena viene caracterizada por tres valores, las coordenadas (X, Y, Z) de P en un sistema de coordenadas con su origen en el orificio de la cámara y el eje

Z a lo largo del eje óptico (Figura 24.1). De lo que disponemos es de la proyección perspectiva del punto en la imagen (*x*,*y*). Esto define el rayo que parte del orificio y sobre el que se encuentra el punto *P*; lo que no conocemos es la distancia. El término «orientación» podemos utilizarlo con dos sentidos:

1. **La orientación del objeto como un todo**. Puede especificarse en términos de rotación tridimensional relacionando su sistema de coordenadas con el de la cámara.
2. **La orientación de la superficie del objeto en *P***. Puede especificarse mediante un vector normal **n**, es decir, un vector que define la dirección que es perpendicular a la superficie. A menudo la orientación de la superficie se expresa usando las variables **inclinación** o *slant* y **ladeo** o *tilt*. Inclinación es el ángulo entre el eje *Z* y **n**. Ladeo es el ángulo entre el eje *X* y la proyección de **n** en el plano de la imagen.

INCLINACIÓN

LADEO

FORMA

Cuando la cámara se mueve con respecto a un objeto, la distancia al objeto y su posición cambian. Lo que se preserva es la **forma** del objeto. Si el objeto es un cubo, este hecho no cambia cuando el objeto se mueve. Los geómetras han intentando formalizar la forma de los objetos durante siglos, y es el concepto básico que la forma es lo que permanece sin cambios bajo un cierto grupo de transformaciones, por ejemplo, combinaciones de rotaciones y traslaciones. La dificultad estriba en encontrar una representación de la forma global del objeto que sea lo suficientemente general como para adaptarse a la amplia variedad de objetos del mundo real (no únicamente formas simples como cilindros, conos y esferas) y que además pueda obtenerse de forma sencilla a partir de una entrada visual. El problema de caracterizar la forma *local* de una superficie se conoce mucho mejor. En esencia, esta caracterización puede realizarse en términos de curvatura: qué cambio sufre la normal a la superficie conforme nos movemos en diferentes direcciones sobre ella. Para un plano, no hay ningún cambio. Para un cilindro, si nos movemos de forma paralela a su eje, no hay cambio, pero en la dirección perpendicular, la normal a la superficie rota en una cantidad inversamente proporcional al radio del cilindro. Todo esto es objeto de estudio de la geometría diferencial.

La forma de un objeto es relevante para algunas tareas de manipulación (por ejemplo, decidir de dónde agarrar un objeto), pero adquiere la mayor importancia en el reconocimiento de objetos, donde la forma geométrica, junto con el color y la textura, proporciona las claves más significativas para permitirnos identificar objetos, clasificar lo que hay en la imagen como un objeto de una clase que ya hemos visto antes, etc.

La cuestión fundamental es la siguiente: dado que durante la proyección perspectiva todos los puntos del mundo tridimensional que se encuentran sobre el rayo que parte del orificio de la cámara se han proyectado sobre el mismo punto de la imagen, ¿cómo recuperamos la información tridimensional? Hay un número de claves disponibles en los estímulos visuales para esta recuperación, incluyendo el **movimiento**, la **estereoscopia binocular**, la **textura**, el **sombreado** y el **contorno**. Cada una de estas claves se basa en unos supuestos básicos sobre las escenas físicas para proporcionar interpretaciones no ambiguas (o casi). Discutimos cada una de estas claves en las cinco subsecciones siguientes.

Movimiento

Hasta ahora hemos considerado una única imagen cada vez. Pero las videocámaras capturan 30 fotogramas por segundo, y las diferencias entre fotogramas pueden ser una importante fuente de información. Si la cámara se mueve respecto a la escena tridimensional, el movimiento aparente resultante en la imagen se llama **flujo óptico**. Este describe la dirección y la velocidad de movimiento de las características *de la imagen* como resultado del movimiento relativo entre el observador y la escena. En la Figura 24.7(a) y (b), mostramos dos fotogramas de un vídeo de un cubo de Rubik girando. En (c) se muestran los vectores del flujo óptico calculados a partir de estas imágenes. El flujo óptico codifica información útil sobre la estructura de la escena. Por ejemplo, cuando los vemos desde un automóvil en movimiento, los objetos lejanos tienen un movimiento aparente mucho menor que los que están más cercanos; por lo tanto, la cantidad de movimiento aparente nos puede decir algo sobre la distancia.

FLUJO ÓPTICO

El campo vectorial del flujo óptico puede representarse a través de sus componentes $v_x(x, y)$ en la dirección x y $v_y(x, y)$ en la dirección y. Para medir el flujo óptico, necesitamos encontrar los puntos correspondientes entre un fotograma y el siguiente. Aprovechamos el hecho de que las zonas de la imagen alrededor de los puntos que se corresponden tienen patrones de intensidad similares. Consideremos un bloque de píxeles centrados en el píxel p, (x_0, y_0) en el momento t_0. Este bloque de píxeles debe compararse con bloques de píxeles centrados en varios píxeles candidatos q_i que se encuentran en $(x_0 + D_x, y_0 + D_y)$ en el momento $t_0 + D_t$. Una posible medida de similitud es la **suma de las diferencias al cuadrado** (SDC):

SUMA DE LAS DIFERENCIAS AL CUADRADO

$$SDC(D_x, D_y) = \sum_{(x,y)} (I(x, y, t) - I(x + D_x, y + D_y, t + D_t))^2$$

Aquí, (x, y) va tomando valores dentro de los píxeles del bloque centrado en (x_0, y_0). Buscamos el valor de (D_x, D_y) que minimice SDC. El flujo óptico en (x_0, y_0) es entonces $(v_x, v_y)=(D_x/D_t, D_y/D_t)$. Otra alternativa es maximizar la **correlación cruzada**:

CORRELACIÓN CRUZADA

$$Correlación(D_x, D_y) = \sum_{(x,y)} (I(x, y, t)\, I(x + D_x, y + D_y, t + D_t)$$

Figura 24.7 (a) Un cubo de Rubik en una mesa rotatoria. (b) El mismo cubo, mostrado 19/30 segundos después. (Cortesía de Richard Szeliski.) (c) Vectores de flujo calculados comparando las dos imágenes de (a) y (b). (Cortesía de Joe Weber y Jitendra Malik.)

La correlación cruzada trabaja mejor cuando hay textura en la escena, y resulta en una ventana que contiene variaciones significativas de brillo entre los píxeles. Si miramos a un muro blanco uniforme, la correlación cruzada será casi la misma para los diferentes puntos q candidatos a ser emparejados, y el algoritmo se reduce a adivinar a ciegas.

EGOMOVIMIENTO

Supongamos que el observador tiene una velocidad traslacional **T** y una velocidad angular ω (que describe de esta manera el movimiento del observador o **egomovimiento**). Podemos derivar una ecuación que relacione las velocidades del observador, el flujo óptico, y las posiciones de los objetos en la escena. Suponiendo que $f = 1$, se sigue que:

$$v_x(x, y) = \left[-\frac{T_x}{Z(x, y)} - \omega_y + \omega_z y\right] - x\left[-\frac{T_z}{Z(x, y)} - \omega_x y + \omega_y x\right]$$

$$v_y(x, y) = \left[-\frac{T_y}{Z(x, y)} - \omega_z x + \omega_x\right] - y\left[-\frac{T_z}{Z(x, y)} - \omega_x y + \omega_y x\right]$$

donde $Z(x,y)$ da la coordenada z del punto de la escena correspondiente al punto en la imagen (x,y).

Se puede entender de forma intuitiva si consideramos el caso de una traslación pura. En este caso, el campo de flujo es

$$v_x(x, y) = \frac{-T_x + xT_z}{Z(x, y)}, \qquad v_y(x, y) = \frac{-T_y + yT_z}{Z(x, y)}$$

FOCO DE EXPANSIÓN

Ahora algunas propiedades interesentes salen a la luz. Ambos componentes del flujo óptico, $v_x(x, y)$ y $v_y(x, y)$, son cero en el punto $x = T_x/T_z$, $y = T_y/T_z$. Este punto se denomina **foco de expansión** del campo de flujo. Supongamos que cambiamos el origen en el plano x-y para que se encuentre en el foco de expansión; entonces las expresiones para el flujo óptico toman una forma particular muy simple. Sean (x', y') las nuevas coordenadas definidas por $x' = x - T_x/T_z$, $y' = y - T_y/T_z$. Entonces

$$v_x(x', y') = \frac{x'T_z}{Z(x', y')}, \qquad v_y(x', y') = \frac{y'T_z}{Z(x', y')}$$

Esta ecuación tiene algunas aplicaciones interesantes. Supongamos que somos una mosca intentando aterrizar sobre un muro y queremos saber el tiempo para tomar contacto a la velocidad actual. Este tiempo viene dado por Z/T_z. Obsérvese que aunque el campo de flujo óptico instantáneo no puede proporcionar la distancia Z o la componente T_z de la velocidad, sí puede proporcionar la razón entre las dos y puede, por lo tanto, utilizarse para controlar la aproximación al aterrizar. Diversos experimentos con moscas reales han demostrado que este es exactamente el mecanismo que ellas usan. Las moscas son los voladores más diestros de entre todos los animales o máquinas, y es interesante constatar que lo hacen con un sistema de visión que tiene una resolución espacial muy pobre (tienen sólo unos 600 receptores frente a los 100 millones de los humanos) pero una resolución temporal espectacular.

Para recuperar la profundidad, debemos hacer uso de múltiples fotogramas. Si una cámara mira hacia un cuerpo rígido, la forma no cambia de un fotograma a otro, y por lo tanto somos capaces de manejar mejor las medidas del flujo óptico que son intrínsecamente ruidosas. Los resultados de un enfoque de este tipo debido a Tomasi y Kanade (1992) se muestran en las Figuras 24.8 y 24.9.

Figura 24.8 (a) Cuatro fotogramas de una secuencia de vídeo en la que la cámara se mueve y gira con respecto al objeto. (b) El primer fotograma de la secuencia, con pequeños cuadrados que señalan las características encontradas por el detector de características. (Cortesía de Carlo Tomasi.)

(a) (b)

Figura 24.9 (a) Reconstrucción tridimensional de las posiciones de las características de la imagen de la Figura 24.8, vista desde arriba. (b) La casa en realidad, tomada desde la misma posición.

Estereoscopia binocular

ESTEREOSCOPIA BINOCULAR

La mayoría de los vertebrados tienen *dos* ojos. Esto es útil en el caso de la pérdida de un ojo por la redundancia que supone, pero también supone una ayuda en otras ocasiones. La mayoría de los animales que sirven de presa tienen ojos a los lados de la cabeza para permitir un campo de visión más amplio. Los depredadores, sin embargo, tienen los ojos en la parte delantera, para permitirles utilizar la **estereoscopia binocular**. La idea es similar a la de paralaje por movimiento[4], excepto en que en vez de utilizar el movimiento a lo largo del tiempo, utilizamos dos (o más) imágenes separadas en el espacio, tal y como las proporcionan los ojos de los humanos situados en la parte delantera de la cara. Puesto que un elemento dado de la escena estará en diferente lugar en relación con el eje z de cada plano de imagen, si superponemos las dos imágenes habrá una **disparidad** en la localización del elemento en las dos imágenes. Podemos verlo en la Figura 24.10, donde el punto más cercano de la pirámide está desplazado a la izquierda en la imagen derecha y a la derecha en la imagen izquierda.

DISPARIDAD

Calculemos la relación geométrica entre la disparidad y la profundidad. En primer lugar consideraremos el caso en el que ambos ojos (o cámaras) están mirando hacia delante con sus ejes ópticos paralelos. La relación de la cámara derecha con la izquierda es solamente una traslación a lo largo del eje x una cantidad b, la línea base. Podemos utilizar las ecuaciones del flujo óptico de la sección anterior para computar las disparidades horizontal y vertical como $H = v_x \Delta t$, $V = v_y \Delta t$, dado que $T_x = b/\Delta t$ y $T_y = T_z = 0$.

Figura 24.10 Idea de la estereoscopia: las diferencias en las posiciones de las cámaras producen vistas bidimensionales de la misma escena tridimensional ligeramente distintas.

[4] *Nota del traductor:* Diferencia entre las posiciones de los objetos en el tiempo. Generalmente utilizada en Astronomía.

Los parámetros rotacionales ω_x, ω_y y ω_z son cero. Se obtiene que $H = b/Z$, $V = 0$. En palabras, la disparidad horizontal es igual a la división de la línea base entre la profundidad, y la disparidad vertical es cero.

ENFOQUE

Bajo condiciones visuales normales, los humanos **enfocamos**; es decir, hay algún punto en la escena en el que los ejes ópticos de los dos ojos intersectan. La Figura 24.11 muestra dos ojos enfocados en un punto P_0, que está a una distancia Z del punto medio entre los dos ojos. Por conveniencia, calcularemos la disparidad *angular*, medida en radianes. La disparidad en el punto enfocado P_0 es cero. Para cualquier otro punto P de la escena que se encuentra a una distancia δZ del anterior, podemos calcular los desplazamientos angulares de las imágenes izquierda y derecha de P, que llamaremos P_L y P_R, respectivamente. Si cada uno de ellos está desplazado un ángulo $\delta\theta/2$ relativo a P_0, entonces el desplazamiento entre P_L y P_R, que es la disparidad de P, es justamente $\delta\theta$. Por simple geometría tenemos

$$\frac{\delta\theta}{\delta Z} = \frac{-b}{Z^2}$$

LÍNEA BASE

En los humanos, b (la **línea base**) es de unos 6 cm. Supongamos que Z es de unos 100 cm. Entonces el menor $\delta\theta$ detectable (correspondiente al tamaño del píxel) es de unos cinco segundos de arco, dando un δZ de 0,4 mm. Para $Z = 30$ cm, tenemos un valor impresionantemente pequeño de $\delta Z = 0{,}036$ mm. Es decir, a una distancia de 30 cm, los humanos podemos discriminar profundidades que difieren tan sólo en 0,036 mm, permitiéndonos enhebrar agujas y cosas así.

Gradientes de textura

TEXTURA

La **textura**, en un lenguaje llano, es la propiedad de las superficies asociada con la calidad táctil que sugieren («textura» tiene la misma raíz que «textil»). En visión por computador, se refiere a un concepto íntimamente relacionado, que es el de un patrón repetido espacialmente sobre una superficie y que puede captarse visualmente. Entre los ejemplos podemos incluir el patrón de las ventanas sobre un edificio, las puntadas de un

Figura 24.11 Relación entre la disparidad y la profundidad en la estereoscopia.

suéter, las manchas de la piel de un leopardo, las briznas de hierba en una pradera, los guijarros en una playa o la multitud en un estadio. A veces la disposición es bastante periódica, como las puntadas del suéter; otras veces, como los guijarros de la playa, la regularidad aparece sólo en un sentido estadístico: la densidad de guijarros es aproximadamente la misma en diferentes partes de la playa.

TEXELS

Lo que acabamos de decir es cierto en el caso *de la escena*. En la imagen, el tamaño aparente, la forma, la separación, etc., de los elementos de la textura (los **texels**) sí que varían, como se ilustra en la Figura 24.12. Las baldosas son idénticas en la escena. Hay dos causas principales para la variación en el tamaño y la forma de las baldosas una vez proyectadas en la imagen:

1. *Diferencias en las distancias de los texels a la cámara*. Recordemos que bajo una proyección perspectiva, los objetos lejanos aparecen más pequeños. El factor de escala es de $1/Z$.
2. *Diferencias en el escorzo*[5] *de los texels*. Esto está relacionado con la orientación de cada texel en relación con la línea de vista desde la cámara. Si el texel es perpendicular a la línea de vista, no hay escorzo. La magnitud del efecto de escorzo es proporcional a cos σ, donde σ es la inclinación del plano del texel.

GRADIENTES DE TEXTURA

Mediante algunos cálculos matemáticos, podemos calcular expresiones para la proporción de cambio de varias características de los texels de la imagen, tales como área, escorzo y densidad. Estos **gradientes de textura** son funciones de la forma de la superficie, así como de su inclinación y ladeo con respecto a la posición del observador.

(a) (b)

Figura 24.12 (a) Una escena que ilustra el gradiente de textura. Suponer que la textura real es uniforme permite recuperar la orientación de la superficie. La orientación calculada se indica superponiedo un círculo y un indicador, transformados como si el círculo se encontrara pintado sobre la superficie en ese punto. (b) Recuperación de la forma a partir de una textura para una superficie curva. (Imágenes cortesía de Jitendra Malik y Ruth Rosenholtz (1994).)

[5] *Nota del traductor:* Acortamiento de las medidas al representarlas sobre el plano de imagen por efecto de la perspectiva.

Para recuperar la forma a partir de la textura, podemos usar un proceso en dos pasos: (a) medir los gradientes de textura; (b) estimar la forma de la superficie, la inclinación y el ladeo que darían lugar a los gradientes de textura medidos. Mostramos los resultados de este proceso en la Figura 24.12.

Sombreado

El sombreado (variación de la intensidad de la luz recibida desde diferentes porciones de una superficie de una escena) viene determinado por la geometría de la escena y las propiedades reflectantes de las superficies. En gráficos por computador, el objetivo es calcular el brillo de la imagen $I(x, y)$, dada la geometría de la escena y las propiedades reflectantes de los objetos de la escena. La visión por computador tiene como objetivo invertir el proceso, esto es, recuperar la geometría y las propiedades reflectantes, dado el brillo de la imagen $I(x, y)$. Se ha comprobado que este proceso es muy difícil de realizar más allá de los casos más sencillos.

Comencemos con una situación en la que podamos, de hecho, resolver la recuperación de la forma a partir del sombreado. Consideremos una superficie Lambertiana iluminada por una fuente de luz puntual lejana. Asumiremos que la superficie está lo suficientemente lejana de la cámara como para que podamos utilizar una proyección ortográfica como aproximación de una proyección perspectiva. El brillo de la imagen es

$$I(x, y) = k\mathbf{n}(x, y) \cdot \mathbf{s}$$

donde k es una constante de escalado, $\mathbf{n}$ es el vector unitario normal a la superficie y $\mathbf{s}$ es el vector unitario en la dirección de la fuente de luz. Puesto que $\mathbf{n}$ y $\mathbf{s}$ son vectores unitarios, su producto escalar es simplemente el coseno del ángulo entre ellos. La forma de la superficie se capta a través de la variación del vector normal $\mathbf{n}$ a lo largo de la superficie. Supongamos que k y $\mathbf{s}$ son conocidos. Nuestro problema es ahora recuperar el vector normal a la superficie $\mathbf{n}(x, y)$ dada la intensidad de la imagen $I(x, y)$.

La primera observación que debemos hacer es que el problema de determinar $\mathbf{n}$, dado el brillo I de un píxel dado (x, y), está indeterminado localmente. Podemos calcular el ángulo que forma $\mathbf{n}$ con el vector de la fuente de luz, pero esto sólo restringe su posición a encontrarse dentro de un cierto cono de direcciones con eje $\mathbf{s}$ y un ángulo en su vértice de $\theta = \cos^{-1}(I/k)$. Para continuar, observemos que $\mathbf{n}$ no puede variar arbitrariamente de un píxel a otro. Se corresponde con el vector normal de una porción de superficie suave y consecuentemente debe variar también de una forma suave; el término técnico para esta restricción es **integrabilidad**. Se han desarrollado varias técnicas diferentes para explotar esta característica. Una es simplemente rescribir $\mathbf{n}$ en términos de las derivadas parciales Z_x y Z_y de la profundidad $Z(x, y)$. El resultado es una ecuación diferencial parcial para Z que puede resolverse para obtener la profundidad $Z(x, y)$, dadas unas condiciones de frontera apropiadas.

INTEGRABILIDAD

Se puede generalizar el enfoque de alguna manera. No es necesario que la superficie sea Lambertiana ni que la fuente de luz sea una fuente puntual. Siempre que podamos computar el **mapa de reflectancia** $R(\mathbf{n})$, que especifica el brillo de una porción de la superficie como una función de su normal $\mathbf{n}$, podemos utilizar en esencia el mismo tipo de técnicas.

MAPA DE REFLECTANCIA

La verdadera dificultad aparece cuando tratamos las reflexiones entre objetos. Si consideramos una escena de interior típica, por ejemplo los objetos de una oficina, las superficies están iluminadas no sólo por las fuentes de luz, sino también por la luz reflejada por otras superficies de la escena que sirven en realidad como fuentes de luz secundarias. Este efecto de iluminación mutua es bastante significativo. El formalismo del mapa de reflectancia falla completamente en esta situación: el brillo de la imagen depende no sólo de la normal a la superficie, sino también de las relaciones espaciales complejas entre las diferentes superficies de la escena.

Está claro que los humanos obtenemos alguna percepción de la forma a través del sombreado, por lo que este continúa siendo un problema interesante a pesar de todas estas dificultades.

Contorno

Cuando miramos los trazos de un dibujo de líneas, como el de la Figura 24.13, tenemos una percepción vívida de la forma y distribución tridimensional. ¿Cómo? Al fin y al cabo, vimos anteriormente que existen infinidad de configuraciones de la escena que pueden llevar a los mismos trazos del dibujo. Observemos que somos capaces incluso de percibir una superficie ladeada o inclinada. Esto puede deberse a una combinación de conocimiento de alto nivel (sobre formas comunes) y restricciones de bajo nivel.

Consideraremos el conocimiento cualitativo disponible de un dibujo de líneas. Como se ha tratado anteriormente, las líneas de un dibujo pueden tener múltiples significados. (*Véase* Figura 24.4 y texto adjunto.) La tarea de evaluación del verdadero significado de cada línea en una imagen se denomina **etiquetado de líneas** y fue una de las primeras tareas estudiadas en visión por computador. Por ahora, trataremos con un modelo simplificado del mundo donde los objetos que no tienen marcas de superficie y las líneas que se deben a discontinuidades de iluminación, como el borde de sombra y brillos es-

ETIQUETADO DE LÍNEAS

Figura 24.13 Un dibujo de línea evocador. (Cortesía de Isha Malik).

peculares, han sido borrados en algún paso del proceso, permitiéndonos limitar nuestra atención a los trazados de un dibujo donde cada línea corresponde a una profundidad o a una discontinuidad en la orientación.

LÍNEA DE SILUETA

Entonces, cada línea puede clasificarse como la proyección de una **línea de silueta** (el lugar geométrico de puntos de la superficie donde la línea de vista es tangente a la superficie) o como un **borde** (discontinuidad de la normal de la superficie). Además, cada borde se puede clasificar como convexo, cóncavo y de oclusión. En el caso de los bordes de oclusión y líneas de silueta, deberemos conocer cuál de las dos superficies que limitan a la curva en el dibujo es la más cercana en la escena. Estas inferencias son representadas asignándole a cada línea una de las seis **etiquetas de línea** posibles, como está ilustrado en la Figura 24.14:

ETIQUETAS DE LÍNEA

1. Las etiquetas «+» y «−» representan bordes convexos y cóncavos, respectivamente. Se asocian con discontinuidades en las normales a la superficie donde ambas superficies que se encuentran en el borde son visibles.
2. Una «←» o una «→» representan un borde convexo de oclusión. Vistos desde la cámara, ambos fragmentos de superficie que se juntan en el borde, se encuentran en el mismo lado, uno ocultando al otro. Conforme uno se mueve en la dirección de la flecha, las superficies quedan a la derecha.
3. Una «←←» o una «→→» representan una línea de silueta. La superficie se curva suavemente para ocultarse ella misma. Conforme uno se mueve en la dirección de las flechas dobles, la superficie se sitúa a la derecha. La línea de vista es tangente a la superficie para todos los puntos de la línea de silueta. Las líneas de silueta se mueven sobre la superficie del objeto conforme cambia el punto de vista.

De las 6^n posibles combinaciones de asignación de etiquetas para las n líneas del dibujo, sólo una pequeña cantidad son físicamente posibles. La determinación de estas asignaciones de etiquetas es el problema del etiquetado de línea. Nótese que este problema únicamente tiene sentido si la etiqueta es la misma en toda la línea. Esto no siempre es cierto, porque la etiqueta puede cambiar a lo largo de una línea en imágenes de objetos curvos. Trataremos sólo objetos poliédricos para evitar este problema.

Huffman (1971) y Clowes (1971) intentaron, independientemente, la primera aproximación sistemática al análisis de escenas poliédricas. Huffman y Clowes limitaron sus

Figura 24.14 Diferentes tipos de etiquetas de línea.

TRIÉDRICOS

análisis a escenas con sólidos **triédricos** opacos (objetos en los que en cada vértice convergen exactamente tres superficies planas). Para escenas con múltiples objetos, descartaron las alineaciones de objetos que podrían suponer una violación del supuesto triédrico, como dos cubos compartiendo un borde común. Las **rupturas** (por ejemplo, «bordes» a través de los cuales los planos tangentes son continuos) tampoco se permiten. Para el mundo triédrico, Huffman y Clowes realizaron una lista exhaustiva de los diferentes tipos de vértices y de las diferentes formas de observarlos bajo un punto de vista general. La condición del punto de vista general asegura esencialmente que si existe un pequeño movimiento del ojo, ninguna de las aristas cambia su carácter. Por ejemplo, esta condición implica que si intersectan tres líneas en la imagen, los bordes correspondientes en la escena deben intersectar también.

RUPTURAS

En la Figura 24.15 se muestran los cuatro modos en los que pueden concurrir tres superficies planas en un vértice. Estos casos han sido construidos tomando un cubo y dividiéndolo en ocho **octantes**. Queremos generar todos los posibles vértices triédricos del centro del cubo rellenándolo con varios octantes. El vértice etiquetado con 1 corresponde a un octante relleno, 3 con tres octantes rellenos, etcétera. Los lectores deben convencerse que estas son, en efecto, *todas* las posibilidades. Por ejemplo, si uno rellena dos octantes en un cubo, uno no puede construir un vértice triédrico válido en el centro. Nótese también que estos cuatro casos corresponden a combinaciones diferentes de bordes cóncavos y convexos que coinciden en el vértice.

OCTANTES

Los tres bordes que se encuentran en el vértice dividen el espacio que les rodea en ocho octantes. Un vértice puede verse desde cualquier octante no ocupado por material sólido. Moviendo el punto de vista dentro de un octante no produce un dibujo con diferentes tipos de unión. El vértice etiquetado como 1 en la Figura 24.15 puede verse desde cualquiera de los siete octantes restantes para producir las etiquetas de unión en la Figura 24.16.

Figura 24.15 Los cuatro tipos de vértices triédricos.

Figura 24.16 Diferentes aspectos del vértice 1 de la Figura 24.15.

Un listado exhaustivo de las diferentes formas en que cada vértice puede observarse da como resultado las posibilidades que se muestran en la Figura 24.17. En la imagen podemos distinguir cuatro tipos de unión diferentes: L-, Y-, flechas, y uniones T. Las uniones L corresponden a dos bordes visibles. Las uniones Y y flechas corresponden a bordes triples (en una unión Y ninguno de los tres ángulos es mayor que 180°). Las uniones T se asocian con la oclusión. Cuando una superficie más cercana y opaca obstaculiza la vista de un borde a mayor distancia, se obtiene un borde continuo con un encuentro en la mitad del borde. Las cuatro etiquetas de la unión T corresponden a la oclusión de cuatro tipos diferentes de bordes.

En la utilización de este diccionario de uniones, para encontrar una etiqueta para el dibujo de líneas, el problema es descubrir qué interpretaciones de unión son globalmente consistentes. La consistencia se fuerza por la regla que indica que a cada línea del di-

Figura 24.17 Conjunto de etiquetas de Huffman-Clowes.

bujo se le debe asignar una y sólo una etiqueta, en toda su longitud. Waltz (1975) propuso un algoritmo para este problema (en realidad, para una versión ampliada con sombras, rupturas y bordes cóncavos separables) el cual fue una de las primeras aplicaciones de satisfacción de restricciones en IA (*véase* Capítulo 5). En la terminología CSP, las variables son las uniones, los valores son las etiquetas de las uniones y las restricciones son que cada línea tiene una única etiqueta. Aunque el problema del etiquetado de líneas para escenas triédricas es NP-completo, en la práctica los algoritmos estándar CSP obtienen buenos resultados.

24.5 Reconocimiento de objetos

La visión nos permite reconocer de forma fiable gente, animales y objetos inanimados. En IA o visión por computador, es habitual utilizar el término *reconocimiento de objetos* para referirse a todas estas capacidades. Esto incluye la determinación de la clase de objetos particulares que aparecen en la imagen (por ejemplo, una cara) así como el reconocimiento específico de objetos (por ejemplo, la cara de Bill Clinton). Las aplicaciones que motivan estas técnicas incluyen las siguientes:

IDENTIFICACIÓN BIOMÉTRICA

- **Identificación biométrica**: las investigaciones criminales y el control de acceso para instalaciones restringidas requieren la habilidad de identificar de forma única a los individuos. Huellas dactilares, escáner de iris y fotografías faciales tienen como resultado imágenes que deben identificar a individuos específicos.

RECUPERACIÓN DE IMÁGENES BASADO EN CONTENIDO

- **Recuperación de imágenes basado en el contenido**: es sencillo encontrar la ubicación en un documento, si existe, de la cadena «gato», cualquier editor de textos proporciona esta capacidad. Ahora, consideramos el problema de encontrar un conjunto de píxeles en una imagen que corresponden a la imagen de un gato. Si uno tuviera esta capacidad, podría recuperar de la imagen hechos como «Bill Clinton y Nelson Mandela juntos», «un patinador en el aire», «la Torre Eiffel de noche», etcétera, sin tener que teclear el título para cada fotografía de una colección. Mientras crezcan las colecciones de imágenes y vídeo, las anotaciones manuales no pueden aumentar.

RECONOCIMIENTO DE ESCRITURA

- **Reconocimiento de escritura**: los ejemplos incluyen firmas, bloques de direcciones en sobres, importe de cheques y entrada con bolígrafo en PDAs.

La visión se utiliza para reconocer no sólo objetos, sino también actividades. Podemos identificar pasos (el caminar de un amigo), expresiones (una sonrisa, una mueca), gestos (una persona haciendo señas con una mano), acciones (saltar, bailar), etcétera. La investigación en actividades de reconocimiento está todavía en su infancia, por esto mismo en esta sección nos centraremos en el reconocimiento de objetos.

El problema del reconocimiento visual de objetos es generalmente sencillo para las personas, pero se ha probado que es muy complejo para los computadores. Uno quiere ser capaz de identificar la cara de una persona a pesar de las variaciones de iluminación, postura respecto a una cámara y expresiones faciales. Cualquiera de estos cambios causa diferencias generales en los valores de brillo de un píxel, así que realizar sencillamente

una comparación de píxeles es poco probable que funcione. Cuando se desea reconocer ejemplos de una categoría, como por ejemplo un «coche», se debe hacer frente a las variaciones dentro de la categoría. Incluso el restringido problema del reconocimiento de los números escritos a mano en los códigos postales resulta ser un desafío.

El aprendizaje supervisado o clasificación de patrones proporciona una plataforma natural para el estudio del reconocimiento de objetos. Dando imágenes de ejemplos positivos («caras») y ejemplos negativos («no caras»), el objetivo es el aprendizaje de una función que pueda hacer corresponder imágenes originales a una de las etiquetas *cara*, *no cara*. Todas las técnicas de los capítulos 18 y 20 pueden ser candidatas: perceptrones multicapa, árboles de decisión, clasificadores por el vecino más cercano, y máquinas núcleo[6], todas han sido aplicadas a los problemas de reconocimiento de objetos. Debemos anotar, sin embargo, que la aplicación de estas técnicas para el reconocimiento de objetos está lejos de ser sencilla.

El primer desafío es la segmentación de la imagen. Cualquier imagen contendrá múltiples objetos, por eso necesitamos en un principio dividirla en grupos de píxeles que corresponden con objetos individuales. Una vez la imagen está dividida en regiones, se pueden introducir estas regiones o conjuntos de regiones en un clasificador para determinar las etiquetas de los objetos. Desafortunadamente, la segmentación de abajo a arriba (*bottom-up*) es un proceso propenso a tener errores, por eso como alternativa se debe intentar buscar grupos de objetos de *arriba abajo* (*top-down*). Esto es, buscar un conjunto de píxeles que se puedan clasificar como una cara, y si se tiene éxito, ¡se ha encontrado un grupo! Las estrategias *top-down* tienen una alta complejidad computacional, porque se necesita examinar ventanas de imagen de varios tamaños, en diferentes ubicaciones, así como compararlas con todas las diferentes hipótesis de objetos. En el presente, muchos sistemas prácticos de reconocimiento de objetos utilizan una estrategia de arriba abajo (*top-down*), aunque esto debe cambiar conforme mejoren las técnicas de abajo arriba (*bottom-up*).

El segundo reto es asegurar que el proceso de reconocimiento es robusto con variaciones de iluminación y postura. Los humanos pueden reconocer objetos a pesar de variaciones considerables en la apariencia exacta, medida por los valores de intensidad de los píxeles. Por ejemplo, podemos reconocer la cara de un amigo bajo diferentes condiciones de iluminación, o desde diferentes ángulos. Veamos un ejemplo sencillo, si consideramos el número 6 escrito a mano. Podemos reconocerlo de diferentes tamaños y en diferentes posiciones en la imagen, a pesar de pequeñas rotaciones de la figura[7].

El punto clave a tener en cuenta es que las transformaciones geométricas como traslación, escalabilidad y rotación, o transformaciones del brillo de la imagen causadas por el movimiento físico de las fuentes de luz, tienen diferente carácter que las variaciones dentro de una categoría, como las que existen entre diferentes rostros humanos, o las formas diferentes de escribir el número 4. Por otro lado, los efectos de las transformaciones geométricas y físicas son sistemáticos y uno puede factorizarlos con un diseño apropiado de los rasgos utilizados para representar los casos de aprendizaje.

Para evitar variaciones en las transformaciones geométricas, una técnica que ha sido probada de forma efectiva es preprocesar la región de la imagen en una posición, esca-

[6] *Nota deltraductor:* Del inglés Kernel Machines.
[7] Una rotación completa no variante no es nunca necesaria ni deseable; se puede confundir un 6 con un 9.

la y orientación estándar. Alternativamente, podemos ignorar la naturaleza casual de las transformaciones geométricas y físicas y pensar en ellas únicamente como otras fuentes de variabilidad para el clasificador. En el conjunto de entrenamiento, es necesario que se proporcione un ejemplo correspondiente a todas estas variaciones, y la esperanza es que el clasificador inducirá un conjunto apropiado de transformaciones sobre la entrada de tal manera que las variaciones son factorizadas.

Veamos ahora algoritmos específicos para el reconocimiento de formas. Para simplificar, nos centraremos en el problema con un escenario de dos dimensiones, con ejemplos de entrenamiento y texto dados en formato de imágenes bidimensionales claras. En dominios como el reconocimiento de letra manuscrita esto es suficiente. Igualmente en los casos de objetos tridimensionales, un estrategia efectiva es representarlos como múltiples vistas bidimensionales (*véase* Figura 24.18) y clasificar nuevos objetos comparándolos con (alguna representación de) las vistas almacenadas.

La anterior sección muestra que existen diferentes señales para extraer información tridimensional sobre la escena. El reconocimiento de objetos está también basado en múltiples señales, identificamos un tigre por la mezcla de colores naranjas y negros, por la textura de rayas y por la forma del cuerpo.

El color y la textura se pueden representar con histogramas o distribuciones de frecuencia empíricas. Dada una imagen ejemplo de un tigre, podemos medir el porcentaje de píxeles en los diferentes colores. Posteriormente, cuando se presenta una imagen desconocida, podemos comparar su histograma de color con los vistos de los ejemplos del tigre. Para analizar texturas, consideramos los histogramas de los resultados de la convolución de una imagen con filtros de diferentes orientaciones y escalas, buscando una coincidencia.

Ha resultado muy difícil el uso de la forma para el reconocimiento de objetos. Hablando en términos generales, existen dos métodos principales: **reconocimiento basa-**

Figura 24.18 Múltiples vistas de dos objetos tridimensionales.

do en el brillo, en el cual se utiliza directamente el valor de brillo de un píxel, y **reconocimiento basado en las características** el cual implica el uso de la disposición espacial de los rasgos extraídos como bordes y puntos clave. Tras tratar cada uno de estos métodos con más detalle, veremos el problema de la **estimación de postura**, esto es, determinar la localización y orientación de los objetos en la escena.

Reconocimiento basado en la intensidad

Dado el subconjunto de píxeles de una imagen que corresponden con un objeto candidato, definimos las características como los valores brutos de la intensidad del píxel en sí mismos. O, en una variante, primero se debe convolucionar la imagen con varios filtros lineales y tratar los valores de los píxeles de la imagen resultante como las características. Este método ha sido un éxito en tareas como el reconocimiento de dígitos manuscritos como se vio en el Apartado 20.7.

Se han utilizado varios métodos estadísticos para desarrollar detectores faciales a partir de bases de datos de imágenes, incluyendo redes neuronales con entradas sin tratar de píxeles, árboles de decisión con características definidas por varios filtros de barra y borde, y modelos de Bayes sencillos con pequeñas características. En la Figura 24.19 se muestran algunos resultados del último método.

Un aspecto negativo del uso de píxeles sin tratar como vector de características es una gran redundancia inherente en esta representación. Considerando dos píxeles cercanos pertenecientes a la mejilla de un rostro; es probable que estén muy relacionados por tener una geometría similar, iluminación, etc. Las técnicas de reducción de datos como análisis del principal componente, pueden ser utilizadas exitosamente para reducir la di-

Figura 24.19 Salida obtenida de un algoritmo de búsqueda de rostros. (Cortesía de Henry Schneiderman y Takeo Kanade.)

mensión del vector de características, permitiendo el reconocimiento de cosas tales como rostros, con mayor velocidad que un método que lo realice con un espacio de mayor dimensión.

Reconocimiento basado en las características

En lugar de utilizar el brillo de píxeles sin tratar como características, podemos detectar y marcar características localizadas espacialmente como regiones y bordes (Apartado 24.3). Hay dos motivos por los cuales se utilizan bordes. El primero es la reducción de datos (hay muchos menos bordes que píxeles en la imagen). El segundo es la permanencia de la iluminación, dentro de un rango adecuado de contraste, los bordes serán detectados aproximadamente en la misma localización, independientemente de la configuración exacta de iluminación. Los bordes son características unidimensionales; pueden utilizarse también características bidimensionales (regiones) y características sin dimensión (puntos). Nótese la diferencia en el tratamiento de la localización espacial en los métodos basados en la intensidad y en las características. En los métodos basados en el brillo, éste está implícitamente codificado como el índice de una componente de un vector de características. En los métodos basados en las características, la localización (x, y) *es* la característica.

La disposición de los bordes es característico de un objeto (esta es una razón por la cual podemos interpretar fácilmente dibujos de línea (Figura 24.13)), aun cuando estas imágenes no ocurren en la realidad. La forma más sencilla de utilizar este conocimiento es con un clasificador por los vecinos más cercanos. Calculamos previamente y almacenamos las configuraciones de los bordes correspondientes a las vistas de objetos conocidos. Dada la configuración de los bordes correspondiente al objeto desconocido en la imagen, podemos determinar la «distancia» a cada miembro de una biblioteca de vistas almacenadas. Un clasificador por el vecino más cercano selecciona la pareja más cercana.

EMPAREJAMIENTO DEFORMABLE

Se han propuesto muchas definiciones diferentes para distancias entre imágenes. Uno de los métodos más interesantes está basado en la idea de **emparejamiento deformable**. D'Arcy Thompson (1917) en su clásico trabajo *Sobre el Crecimiento y la Forma*[8] observó que figuras relacionadas pero no idénticas podían ser deformadas a veces hacia un alineamiento utilizando transformaciones simples coordinadas[9]. En este paradigma, tenemos una idea de la semejanza de la figura en un proceso de tres pasos: (1) Resolver el problema de correspondencia entre las dos figuras, (2) utilizar las correspondencias para estimar una transformación de alineación, y (3) calcular la distancia entre las dos figuras como la suma de los errores de emparejamiento entre puntos correspondientes, conjuntamente con un término que mida la magnitud de la transformación de alineación.

Representamos la figura como un conjunto discreto de puntos muestreados de los contornos internos o externos de la figura. Estos pueden ser obtenidos como emplazamientos de los píxeles de borde encontrados por un detector de bordes, dándonos un con-

[8] *Nota del traductor:* título original: *On Growth and Form*.
[9] En los gráficos por computador actuales, esta idea es conocida como ***morphing***.

junto $\{p_1, p_2, \ldots, p_N\}$ de N puntos. La Figura 24.20 (a) y (b) muestra puntos de ejemplo para dos figuras.

A continuación consideramos un punto de muestra particular p_i, conjuntamente con el conjunto de vectores originados desde ese punto al resto de puntos de muestra de la figura. Estos vectores expresan la configuración de la figura completa relativa al punto de referencia. Lo que nos conduce a la siguiente idea: asociar con cada punto de muestra un descriptor, el **contexto de la figura**, el cual describe toda la disposición del resto de la figura respecto a este punto. De forma más precisa, el contexto de la figura de p_i es un histograma espacial h_i de coordenadas relativas $p_k - p_i$ de los restantes $N - 1$ puntos p_k. Un sistema de coordenadas polares logarítmicas se utiliza para definir secciones asegurando que el descriptor es más sensible a las diferencias en píxeles cercanos. Se muestra un ejemplo en la Figura 24.20(c).

CONTEXTO DE LA FIGURA

Figura 24.20 Cálculo y emparejamiento de los contextos de la figura. (a, b) Puntos muestra de borde de dos figuras. (c) Diagrama de los histogramas de secciones polares logarítmicas utilizados en el cálculo de los contextos de la figura. Utilizamos cinco secciones para *log r* y 12 secciones para θ. (d-f) Ejemplos de contextos de figura para referenciar muestras marcadas por los puntos ○, ◇, ◁ en (a, b). Cada contexto de figura es un histograma polar de las coordenadas del resto de los puntos medidos utilizando el punto de referencia como el origen. (Las celdas oscuras significan más puntos en la sección.) Nótese la similitud visual de los contextos de figura entre ○ y ◇, los cuales han sido calculados para puntos relativamente parecidos en las dos figuras. Por contraste, el contexto de figura para ◁ es bastante diferente. (g) Correspondencias entre (a) y (b) encontradas utilizando emparejamiento bipartito, con coste definido por la distancia χ^2 entre histogramas.

Nótese que la invarianza a la traslación es intrínseca a la definición del contexto de la figura ya que todas las medidas son tomadas respecto a los puntos del objeto. Para lograr invarianza de escala, todas las distancias radiales son normalizadas por la distancia media entre pares de puntos.

Los contextos de la figura nos permiten resolver el problema de correspondencia entre dos figuras similares pero no idénticas, como lo visto en la Figura 24.20(a) y (b). Los contextos de la figura serán diferentes para diferentes puntos en una única figura S, mientras que puntos coincidentes (homólogos) en figuras similares S y S' tenderán a tener unos contextos de figura similares. Podemos resolver el problema de encontrar puntos correspondientes entre dos figuras, encontrando parejas que tengan contextos de figura similares.

Más detalladamente, consideramos un punto p_i en la primera figura y un punto q_j en la segunda figura. Suponemos que $C_{ij} = C(p_i, q_j)$ denota el coste de emparejar los dos puntos. Como los contextos de figura son distribuciones representadas como histogramas, es natural utilizar la distancia x^2:

$$C_{ij} = \frac{1}{2} \sum_{k=1}^{K} \frac{[hi(k) - h_j(k)]^2}{hi(k) + h_j(k)}$$

donde $h_i(k)$ y $h_j(k)$ indican que el k-ésimo cubo de los histogramas normalizados en p_i y q_j. Dado el conjunto de los costes C_{ij} entre todos los pares de puntos i en la primera figura y j en la segunda figura queremos minimizar el coste total de emparejamiento sujeto a la limitación de que el emparejamiento sea uno a uno. Esto es una instancia del problema del **emparejamiento ponderado**, el cual puede ser resuelto en tiempo $O(N^3)$ utilizando el algoritmo Húngaro.

Dadas las correspondencias de puntos de muestra, la correspondencia se puede extender a la figura completa estimando una transformación de alineamiento que empareja una figura sobre la otra. La utilización de las curvas *spline* como formalización es especialmente efectiva. Una vez las figuras están alineadas, calcular puntuaciones similares es relativamente sencillo. La distancia entre dos figuras puede ser definida como una suma ponderada de las distancias de los contextos de la figura entre puntos correspondientes y la energía asociada con la *spline*. Dada esta medida de la distancia, se puede utilizar un simple clasificador por el vecino más cercano para resolver el problema del reconocimiento. La excelente mejora de este método para la clasificación de dígitos escritos a mano ha sido descrita en el Capítulo 20.

Estimación de postura

Además de determinar qué es un objeto, también nos interesa determinar su postura, esto es, su posición y orientación respecto al observador. Por ejemplo, en una tarea de manipulación industrial, el brazo del robot no puede tomar un objeto hasta que la postura sea conocida. En el caso de objetos rígidos, tanto tridimensionales como bidimensionales, este problema tiene una solución sencilla y bien definida basada en el **método de alineación**, el cual desarrollaremos a continuación.

MÉTODO DE ALINEACIÓN

El objeto está representado por M características o puntos distinguidos $m_1, m_2, \ldots, m_M$ en un espacio tridimensional, quizá los vértices de un objeto poliédrico. Estos son

medidos en algún sistema de coordenadas natural para el objeto. Los puntos están sometidos a una rotación tridimensional desconocida **R**, seguida de una traslación por una cantidad desconocida **t** y posteriormente una proyección para dar lugar a una serie de puntos de características de la imagen $p_1, p_2, \ldots, p_N$ en el plano de la imagen. En general, $N \neq M$, porque algunos de los puntos pueden estar ocultos, y el detector de características podría pasar por alto algunas características (o inventar algunas falsas a causa del ruido). Podemos expresar esto como

$$p_i = \Pi(\mathbf{R}m_i + \mathbf{t}) == Q(m_i)$$

para un punto modelo tridimensional m_i, y el punto de la imagen correspondiente p_i. **R** es una matriz de rotación, **t** es una traslación, y Π denota la proyección perspectiva o una de sus aproximaciones, como proyecciones ortográficas a escala. El resultado neto es una transformación Q que lleva a un punto modelo m_i, en alineación con el punto de la imagen p_i. Aunque no conocemos Q inicialmente, sabemos (para objetos rígidos) que Q debe ser la *misma* para todos los puntos del modelo.

Se puede resolver para Q, dadas las coordenadas tridimensionales de tres puntos del modelo y sus proyecciones bidimensionales. Intuitivamente sería como sigue: se pueden expresar ecuaciones relacionando las coordenadas de p_i con las de m_i. En estas ecuaciones, las cantidades desconocidas corresponden a los parámetros de la matriz de rotación **R** y el vector de traslación **t**. Si tenemos suficientes ecuaciones, debemos ser capaces de resolver Q. No veremos ninguna prueba; simplemente plantearemos el siguiente resultado:

> Dados tres puntos no colineales m_1, m_2, y m_3 en el modelo, y sus proyecciones ortográficas a escala p_1, p_2 y p_3 en el plano de la imagen, existen exactamente dos transformaciones desde un sistema de coordenadas del modelo tridimensional, a un sistema de coordenadas de la imagen bidimensional.

Estas transformaciones están relacionadas por un reflejo sobre el plano de la imagen y pueden ser calculadas con una sencilla solución de forma cerrada. Si pudiéramos identificar las características del modelo que corresponden con tres características de la imagen, podríamos calcular Q, la postura del objeto. En la sección anterior, vimos una técnica para determinar correspondencias utilizando el emparejamiento por contexto de la figura. Si el objeto tiene las esquinas bien definidas u otros puntos de interés, entonces una técnica sencilla es apropiada. La idea es generar y probar. Debemos suponer una correspondencia inicial de tres imágenes con tres modelos y utilizar la función ENCONTRAR_TRANSFORMADA para obtener una hipótesis de Q. Si la correspondencia supuesta era correcta, entonces Q será correcta, y aplicada al resto de puntos del modelo, se obtendrá una predicción de los puntos de la imagen. Si la correspondencia supuesta era incorrecta, entonces Q será incorrecta, y aplicada a los puntos del modelo restantes, no predecirán los puntos de la imagen.

Es la base del algoritmo ALINEACIÓN mostrado en la Figura 24.21.El algoritmo halla la postura de un modelo dado, o devuelve un error. La complejidad temporal en el peor caso del algoritmo es proporcional al número de combinaciones de tres modelos y tres imágenes, o $\binom{N}{3}\binom{M}{3}$, veces el coste de verificar cada combinación. El coste de verificación es $M \log N$, ya que debemos predecir la posición de la imagen por cada punto

```
función ALINEAMIENTO(imagen, modelo) devuelve una solución o error
  entradas: imagen, lista de puntos de rasgos de la imagen
            modelo, lista de puntos de rasgos del modelo

  para cada p1, p2, p3 en TRIPLETAS(imagen) hacer
      para cada m1, m2, m3 en TRIPLETAS(modelo) hacer
          Q ← ENCONTRAR_TRANSFORMADA(p1, p2, p3, m1, m2, m3)
          si proyección según Q explica la imagen entonces
              devolver Q
  devolver error
```

Figura 24.21 Una descripción informal del algoritmo de alineamiento.

Figura 24.22 (a) Esquinas encontradas en la imagen de la grapadora. (b) Reconstrucción hipotética superpuesta en la imagen original. (Cortesía de Clark Olson.)

del modelo M, y hallar la distancia al punto de imagen más cercano, una operación log N es una estructura de datos apropiada si los puntos de la imagen están organizados. Por lo tanto, la complejidad en el peor caso del algoritmo de alineación es $O(M^4 N^3 \log N)$, donde M y N son el número de puntos del modelo y de la imagen, respectivamente. Las técnicas basadas en el agrupamiento de posturas en combinación con la aleatorización reducen la complejidad a $O(M N^3)$. Los resultados de la aplicación de este algoritmo a la imagen de la grapadora se muestran en la Figura 24.22.

24.6 Empleo de la visión para la manipulación y navegación

Uno de los principales usos de la visión es proporcionar información para la manipulación de objetos (tomarlos, sujetarlos, girarlos, etc.) y para la navegación evitando obstáculos. La capacidad de utilizar la visión para estos propósitos está presente en los sis-

temas visuales animales más primitivos. En muchos casos, el sistema visual es mínimo, en el sentido de que extrae a partir del campo de visión disponible sólo la información que el animal necesita para informar de su comportamiento. Es muy probable que los sistemas de visión modernos evolucionaran de los primeros organismos primitivos que utilizaban un punto fotosensible en un extremo para orientarse hacia (o lejos de) la luz. Vimos en el Apartado 24.4 que las moscas utilizan un sistema de detección simple del flujo óptico para posarse sobre las paredes. En un estudio clásico, *Lo que el ojo de la rana le dice al cerebro de la rana*[10] (Lettvin *et al*.,1959), observó en una rana que «Se moriría de hambre rodeada de comida si ésta no se moviera. Su posibilidad de comer viene determinada por el tamaño y movimiento».

Los sistemas de visión por computador se utilizan en «organismos» llamados robots. Consideraremos un caso particular de robot: un vehículo automático conducido en una autopista. (*Véase* Figura 24.23.) En un principio, analizaremos la tarea; posteriormente, identificaremos los algoritmos de visión que proveerán la información necesaria para llevar a cabo de forma satisfactoria estas tareas. Las tareas que el conductor debe afrontar son las siguientes:

1. Control lateral: garantiza que el vehículo permanece de forma segura en el carril o cambie de carril suavemente cuando es requerido.
2. Control longitudinal: asegura que existe una distancia de seguridad con el vehículo de enfrente.

Figura 24.23 Imagen de una carretera tomada desde una cámara en el interior de un coche. Las barras blancas horizontales indican las ventanas de búsqueda en las que el controlador busca los marcadores del carril. La baja calidad de la imagen no es atípica en vídeos en escala de grises de baja resolución.

[10] *Nota del traductor:* título original: *What the Frog's Eye Tells the Frog's Brain*.

3. Evitar obstáculos: controlar vehículos en los carriles vecinos y estar preparado para realizar maniobras evasivas si alguno de ellos decide cambiar de carril.

El problema para el conductor es producir las acciones adecuadas de dirección, aceleración y frenado para realizar de la mejor forma posible estas tareas.

Para el control lateral, se necesita mantener una representación de la posición y orientación del coche relativa al carril. En la imagen mostrada en la Figura 24.23, podemos utilizar algoritmos de detección de bordes para encontrar bordes correspondientes a los segmentos de las líneas para marcar los carriles. Entonces, podemos asociar curvas suaves con estos bordes. Los parámetros de estas curvas contienen información sobre la posición lateral del coche, la dirección a la que se dirige relativa al carril, y la curvatura del carril. Esta información, junto con información de la dinámica del coche, es todo lo que necesita el sistema de control de dirección. Nótese también que de cada fotograma al siguiente, existe un cambio muy pequeño en la posición de la proyección del carril en la imagen, se puede conocer *dónde* buscar las marcas de carriles en la imagen, en la figura, necesitamos sólo buscar en las áreas marcadas por barras paralelas blancas.

Para el control longitudinal, se necesita conocer las distancias relativas a los vehículos frontales. Esto se puede llevar a cabo con visión estereoscópica binocular o flujo óptico. Ambas aproximaciones se pueden simplificar utilizando las restricciones de dominio derivadas del hecho de que la conducción se realiza en una superficie plana. Utilizando estas técnicas, coches controlados por visión pueden conducirse a velocidades altas durante distancias largas.

El ejemplo de la conducción deja muy claro un punto: *para tareas específicas, no es necesario recopilar toda la información, que en un principio, puede ser recopilada de una imagen*. No es necesario recopilar la forma exacta de cada vehículo, resolver la forma a partir de la textura de la superficie del césped adyacente a la autopista, etc. La tarea requiere únicamente cierta clase de información, puede ganarse una considerable velocidad computacional y robustez recopilando únicamente esta información y explotando al máximo las restricciones del dominio. Nuestro objetivo a la hora de tratar los métodos generales en la sección anterior es debido a que forman la teoría básica, con la cual cada uno puede especializarse según las necesidades de sus tareas particulares.

24.7 Resumen

Aunque la percepción parece ser una actividad sin esfuerzo para los humanos, requiere una considerable cantidad de cálculos sofisticados. La visión tiene como objetivo extraer información necesaria para tareas tales como manipulación, navegación y reconocimiento de objetos.

- El proceso de **formación de imágenes** es bien conocido en sus aspectos geométricos y físicos. Dada una descripción de una escena tridimensional, podemos generar fácilmente una imagen de ésta desde la posición arbitraria de una cámara (los problemas de gráficos). Invertir el proceso para a partir de una imagen obtener una descripción de la escena, es más difícil.

- Para extraer la información necesaria para la tarea de manipulación, navegación, y reconocimiento, se deben generar representaciones intermedias. Los algoritmos de bajo nivel para el **procesamiento de imágenes** por visión extraen características primitivas de la imagen, como bordes y regiones.
- Existen algunas señales en la imagen que permiten obtener información tridimensional de la escena: movimiento, visión estereoscópica, textura, sombreado y análisis de contornos. Cada una de estas señales se basan en supuestos de contexto sobre escenas físicas para proporcionar interpretaciones sin ambigüedades.
- El reconocimiento de objetos en su totalidad es un problema muy difícil. Tratamos métodos basados en el brillo y basados en características. Y se presentó un algoritmo sencillo para la estimación de la postura. Aunque existen otras posibilidades.

Notas bibliográficas e históricas

Los esfuerzos sistemáticos para entender la visión humana se remontan a tiempos muy antiguos. Euclides (300 a.C.) escribió sobre perspectiva natural: el planeamiento que asocia, con cada punto P del mundo tridimensional, la dirección de la línea OP uniendo el centro de la proyección O con el punto P. Era bien entendido en la noción de paralaje por movimiento. La comprensión matemática de la proyección en perspectiva, esta vez en el contexto de proyección sobre superficies planas, tuvo su siguiente avance significativo en la Italia Renacentista del siglo XV. Es a Brunelleschi (1413) a quien se le atribuye la creación de los primeros dibujos basados en proyecciones geométricamente correctas de una escena tridimensional. En 1435, Alberti codificó las reglas e inspiró generaciones de artistas cuyos logros artísticos todavía nos asombran. Leonardo da Vinci y Albrecht Dürer fueron especialmente notables por su desarrollo de la ciencia de la perspectiva, como lo llamaron en esa época. Las descripciones de Leonardo a finales del siglo XV de la interacción de luz y sombra (claroscuro), las regiones de sombra y penumbra, y la perspectiva aérea merecen todavía ser leídas (Kemp, 1989).

Aunque la perspectiva era conocida por los griegos, éstos estaban confundidos curiosamente por el papel que desempeñaban los ojos en la visión. Aristóteles pensaba en los ojos como un dispositivo emisor de rayos, como los modernos buscadores láser. Esta visión errónea terminó con el trabajo de los científicos árabes, como Alhazen, en el siglo X. Siguió por el desarrollo de una serie de cámaras. Estas cámaras consistían en habitaciones (*camera* es la palabra latina para «habitación») donde la luz entraba por un pequeño agujero en una de las paredes, para proyectar una imagen del mundo exterior en la pared opuesta. Por supuesto, en todas estas cámaras, la imagen estaba invertida, lo que causó una interminable confusión. Si se pensaba en el ojo como un dispositivo de obtención de imágenes, ¿cómo vemos de forma correcta? Este enigma hizo que se ejercitaran las mentes más brillantes de la era (incluido Leonardo). Esto llevó al trabajo de Kepler y Descartes a resolver la cuestión. Descartes colocó un ojo al cual se le había quitado la cutícula opaca en un agujero de una contraventana. El resultado fue una imagen invertida formada en un trozo de papel situado en la retina. Mientras

la imagen de la retina es en verdad invertida, esto no causa un problema porque el cerebro interpreta la imagen de forma correcta. En la jerga moderna, simplemente se debe acceder a la estructura de datos de forma correcta.

Los avances más importantes que siguieron en el estudio de la visión se dieron en el siglo XIX. El trabajo del Helmholtz y Wundt, descrito en el Capítulo 1, estableció la experimentación psicofísica como una disciplina de rigor científico. Gracias al trabajo de Young, Maxwell y Helmholtz, se estableció la teoría tricromática de la visión de los colores. Se demostró gracias al estereoscopio inventado por Wheatstone (1838) que los humanos pueden percibir la profundidad si las imágenes presentadas en el ojo izquierdo y derecho tienen pequeñas diferencias. El dispositivo se volvió popular en las salas y salones de toda Europa. El concepto básico de la visión estereoscópica binocular (dos imágenes de una escena tomadas desde puntos de vista ligeramente diferentes aporta información suficiente para obtener una reconstrucción tridimensional de la escena), fue aprovechada en el campo de la fotogrametría. Se obtuvieron resultados matemáticos clave; por ejemplo, Kruppa (1913) probó que, dadas dos vistas de cinco puntos distintos, se podía construir la rotación y traslación entre las dos posiciones de la cámara así como la profundidad de la escena (hasta un factor de escala). Aunque la geometría de la visión estereoscópica era entendida desde hacía largo tiempo, el problema de la correspondencia en fotogrametría solía ser resuelto intentando establecer una correspondencia entre los puntos. La sorprendente capacidad de los humanos para resolver el problema de la correspondencia fue ilustrada cuando Julesz (1971) inventó el estereograma de puntos aleatorios. Tanto en visión por computador como en fotogrametría, se dedicaron muchos esfuerzos para resolver el problema de la correspondencia en las décadas de 1970 y 1980.

La segunda mitad del siglo XIX fue el período más importante en que se fundamentó el estudio psicofísico de la visión humana. En la primera mitad del siglo XX, los resultados más importantes de las investigaciones en visión fueron obtenidos por la escuela de psicología Gestalt, encabezado por Max Wertheimer. Con el eslogan «el todo es diferente de la suma de las partes», promovieron la idea de que las formas completas, en lugar de componentes tales como bordes, serían las unidades primarias de la percepción.

El período que siguió a la Segunda Guerra Mundial fue marcado por unas actividades renovadas. Los más importantes fueron los trabajos de J. J. Gibson (1950, 1979), el cual indicó la importancia del flujo óptico, así como los gradientes de textura para la estimación de las variables ambientales, como la inclinación y ladeo de una superficie. Volvió a enfatizar la importancia del estímulo y su riqueza. Gibson, Olum y Rosenblatt (1955) indicaron que el campo de flujo óptico contenía suficiente información para determinar el egomovimiento del observador relativo al entorno. La comunidad de la visión por computador desarrolló principalmente en las décadas de 1980 y 1990 el trabajo en el área anterior y en el (matemáticamente equivalente) área de la deducción de la estructura por el movimiento. El trabajo de Koenderink y Van Doorn (1975), Ullman (1979) y Longuet-Higgins (1981) siguió con esta actividad. Las primeras preocupaciones sobre la estabilidad de la estructura del movimiento fueron aliviadas por el trabajo de Tomasi y Kanade (1992) los cuales mostraron que con el uso de múltiples fotogramas, y la amplia línea resultante básica, la forma podía ser reconstruida de forma precisa.

Chan *et al.* (1998) describe el asombroso aparato visual de la mosca, la cual tiene una agudeza visual temporal diez veces mayor que la de los humanos. Esto es, una mosca puede ver una película proyectada a más de 300 fotogramas por segundo y reconocer los fotogramas individuales.

Una innovación conceptual introducida en la década de 1990 fue el estudio de la estructura proyectiva del movimiento. En esta área no es necesaria la calibración de la cámara, como fue mostrado por Faugeras (1992). Este descubrimiento está relacionado con el uso de invariantes geométricas en el reconocimiento de objetos, como fue estudiado por Mundy y Zisserman (1992), y el desarrollo de la estructura afín del movimiento por Koenderink y Van Doorn (1991). En la década de 1990, con un incremento del almacenamiento y velocidad de cálculo, y la disponibilidad general de vídeo digital, el análisis del movimiento encontró nuevas aplicaciones. Construir modelos geométricos de escenas del mundo real para ser renderizados por técnicas de computación gráfica resultó muy popular, conducidos por algoritmos de reconstrucción como el desarrollado por Debevec, Taylor y Malik (1996). El libro de Hartley y Zisserman (2000) y Faugeras *et al.* (2001) proporciona un tratamiento completo de la geometría de múltiples vistas.

En visión por computador, los primeros trabajos sobre la deducción de la forma a partir de la textura corresponden a Bajscy y Liebermann (1976) y Stevens (1981). Mientras este trabajo era para superficies planas, un análisis completo para superficies curvas corresponde a Garding (1992) y Malik y Rosenholtz (1997).

En la comunidad dedicada a la visión por computador, deducir la forma a partir de la textura fue estudiado primeramente por Berthold Horn (1970). Horn y Brooks (1989) presentaron un extenso estudio de los principales artículos en esta área. Este marco realiza una serie de suposiciones de simplificación, el más crítico es aquel en el que se ignora el efecto de la iluminación mutua. La importancia de la iluminación mutua fue bien estudiada por la comunidad de gráficos por computador, donde el rastreo de rayos y radiosidad fueron desarrollados precisamente para considerar la iluminación mutua. Una crítica teórica y empírica se puede encontrar en Forsyth y Zisserman (1991).

En el área de la deducción de la forma a partir del contorno, después de las contribuciones clave iniciales de Huffman (1971) y Clowes (1971), Mackworth (1973) y Sugihara (1984) completaron el análisis para objetos poliédricos. Malik (1987) desarrolló un esquema de etiquetado para objetos curvados lisos fragmentados. Kirousis y Papadimitriou (1988) mostraron que el etiquetado de líneas para escenas triédricas es NP-completo.

Entender los eventos visuales en la proyección de objetos curvados lisos requiere una interacción entre la geometría diferencial y la teoría de la singularidad. El mejor estudio es *Solid Shape* de Koenderink (1990).

El trabajo más relevante en el reconocimiento de objetos tridimensionales fue la tesis de Roberts (1963) presentada ante el MIT. Es considerada como la primera tesis doctoral en visión por computador e introduce algunas ideas clave, como la detección de bordes y emparejamiento basado en modelos. La detección de bordes de Canny fue introducido por Canny (1986). La idea de alineación, también introducida por primera vez por Roberts, resurgió en la década de 1980 en el trabajo de Lowe (1987) y Huttenlocher y Ullman (1990). Mejoras significativas en la eficiencia de la estimación de la postura

CILINDROS GENERALIZADOS

por alineación fueron obtenidas por Olson (1994). Otra nueva rama en la investigación del reconocimiento de objetos 3D fue la aproximación basada en la idea de la descripción de formas en términos de primitivas volumétricas, con **cilindros generalizados**, introducido por Tom Binford (1971), resultando muy popular.

Mientras que la investigación en visión por computador de reconocimiento de objetos se enfocó en cuestiones surgidas de la proyección de objetos tridimensionales en imágenes bidimensionales, hubo una tradición paralela en la comunidad dedicada al reconocimiento de patrones que vio el problema como una clasificación de patrones. Los ejemplos que motivaron esto eran de dominios como reconocimiento óptico de caracteres y reconocimiento de códigos postales manuscritos donde la principal preocupación es el aprendizaje de las variaciones típicas características de una clase de objetos y separar éstas de otras clases. *Véase* LeCun *et al.* (1995) para una comparación de métodos. Otro trabajo de reconocimiento de objetos es el de Sirovitch y Kirby (1987) y de Viola y Jones (2002) para el reconocimiento de rostros. Belongie *et al.* (2002) describen el método de contexto de forma. Dickmanns y Zapp (1987) demostraron en un principio la conducción de un coche controlado por visión en autopistas a gran velocidad; Pomerleau (1993) logró un funcionamiento similar utilizando una red neuronal.

Vision Science: Photons to Phenomenology de Stephen Palmer (1999) proporciona el tratamiento más completo de la visión humana; los libros *Eye, Brain and Vision* de David Hubel (1988) y *Perception* de Irvin Rock (1984) son cortas introducciones centradas en neurofisiología y percepción respectivamente.

En el campo de la visión por computador, el libro más completo disponible hoy en día es *Computer Vision: a Modern Approach* de David Forsyth y Jean Ponce. Se pueden encontrar escritos cortos en los libros de Nalwa (1993) y Trucco y Verri (1998). *Robot Vision* (Horn, 1986) y *Three-Dimensional Computer Vision* (Faugeras, 1993) son dos libros antiguos pero útiles hoy en día, cada uno especializado en un conjunto de temas. El libro de David Marr, *Vision* (Marr, 1982) representa un papel importante en la conexión de la visión por computador a las áreas tradicionales de visión biológica (psicofísica y neurobiología). Dos de las principales publicaciones para visión por computador son el IEEE *Transactions on Pattern Analysis and Machine Intelligence* y el *International Journal of Computer Vision*. Entre las conferencias de visión por computador están ICCV (*International Conference on Computer Vision*), CVPR (*Computer Vision and Pattern Recognition*) y ECCV (*European Conference on Computer Vision*).

EJERCICIOS

24.1 En la sombra de un árbol con un denso y frondoso follaje, es posible observar cierta cantidad de puntos de luz. Sorprendentemente, todos parece que sean circulares. ¿Por qué? Después de todo, los huecos entre las hojas a través de los cuales pasa el sol, no es probable que sean circulares.

24.2 Identifique el dibujo de líneas de la Figura 24.24, suponiendo que los bordes exteriores han sido etiquetados como ocultos y que todos los vértices son triédricos. Realícelo mediante un algoritmo de vuelta atrás (*backtracking*) que examine los vértices en

Figura 24.24 Dibujo para ser etiquetado, en el que todos sus vértices son triédricos.

el orden A, B, C y D, optando en cada paso por una elección que sea consistente con las uniones y bordes etiquetados anteriormente. Pruébelo con el orden B, D, A y C.

24.3 Considere un cilindro infinitamente largo de radio r orientado a lo largo del eje y. El cilindro tiene una superficie Lambertiana vista desde una cámara a lo largo de la parte positiva del eje z. ¿Qué esperaría observar en la imagen si el cilindro está iluminado por una fuente de un punto situada en el infinito en la parte positiva del eje x? Explique su respuesta dibujando los contornos con la misma iluminación en la imagen proyectada. ¿Están estos contornos uniformemente situados?

24.4 Los bordes de una imagen pueden corresponder con varios eventos en una escena. Considere la cubierta de este libro, suponga que es un dibujo de una escena tridimensional real. Identifique 10 bordes de diferente brillo en la imagen, y para cada uno, decida si corresponde con una discontinuidad en (a) profundidad, (b) superficie normal o (d) iluminación.

24.5 Demuestre que la convolución con una función f dada se conmuta con la diferenciación; esto es, demuestre que $(f * g)' = f * g'$.

24.6 Se está considerando un sistema estereoscópico para realizar mapas de un terreno. Consistirá en dos cámaras CCD, cada una tiene 512×512 píxeles en un sensor cuadrado de 10 cm $\times$ 10 cm. Las lentes utilizadas tienen una longitud focal de 16 cm, con el foco fijo en el infinito. Para puntos correspondientes (u_1, v_1) en la imagen izquierda y (u_2, v_2) en la imagen de la derecha, $v_1 = v_2$ porque el eje x en los planos de las dos imágenes son paralelos a las líneas epipolares. Los ejes ópticos de las dos cámaras son paralelos. La línea base entre las dos cámaras es de un metro.

a) Si el rango más cercano que va a ser medido es de 16 metros, ¿cuál es la máxima disparidad que se puede dar (en píxeles)?

b) ¿Cuál es el rango de resolución a los 16 metros, debido al espaciamiento de los píxeles?
c) ¿Qué rango corresponde a una disparidad de un píxel?

24.7 Suponga que deseamos utilizar el algoritmo de alineación en una situación industrial en la que piezas planas se mueven por una cinta transportadora y son fotografiadas por una cámara situada verticalmente por encima de la cinta. La postura de las partes es especificada por tres variables, una para la rotación y dos para la posición bidimensional. Esto simplifica el problema y la función ENCONTRAR-TRANSFORMADA necesita únicamente dos pares de imágenes correspondientes y rasgos del modelo para determinar la postura. Determine la complejidad en el peor caso del procedimiento de alineación.

24.8 La Figura 24.25 muestra dos cámaras en X e Y observando una escena. Dibuje la imagen vista por cada cámara, suponiendo que todos los puntos fijados estén en el mismo plano horizontal. ¿Qué conclusión se puede obtener de estas dos imágenes sobre las distancias relativas de los puntos A, B, C, D y E desde la línea base de la cámara, y con qué base?

24.9 Determine cuál de las siguientes sentencias son verdaderas y cuáles son falsas:

a) Buscar puntos correspondientes en imágenes en estéreo es la fase más sencilla del proceso de búsqueda de la profundidad en estéreo.
b) La obtención de la forma a partir de la textura se puede realizar proyectando una cuadrícula de líneas de luz en la escena.
c) El esquema de etiquetado de Huffman-Clowes puede resolver todos los objetos poliédricos.
d) En los dibujos de líneas de objetos curvos, la etiqueta de la línea puede cambiar de un extremo de la línea a otro.
e) En vistas estéreo de la misma escena, se puede obtener mayor precisión en los cálculos de profundidad si las posiciones de las dos cámaras están más separadas.

Figura 24.25 Vista superior de un sistema de visión con dos cámaras observando una botella con una pared detrás.

f) Líneas con la misma longitud en la escena siempre proyectan iguales longitudes en la imagen.

g) Líneas rectas en la imagen, corresponden necesariamente con líneas rectas en la escena.

24.10 La Figura 24.23 está tomada desde el punto de vista de un coche en el carril de salida de una autopista. Se pueden observar dos coches en el carril inmediato de la izquierda. ¿Qué razones tiene que tener el observador para decidir que uno esta más cerca que el otro?

25 Robótica

Donde los agentes están dotados de efectores físicos para hacer sus travesuras.

25.1 Introducción

ROBOTS

EFECTOR

SENSOR

Los **robots** son agentes físicos que realizan tareas mediante la manipulación física del mundo. Para realizar dichas tareas están equipados con **efectores** como piernas, ruedas, articulaciones y pinzas. Los efectores tienen un único propósito: transmitir fuerzas físicas al entorno[1]. Los robots también están equipados con **sensores**, que les permiten percibir el entorno. Hoy en día, los robots utilizan diversos tipos de sensores, incluyendo cámaras y ultrasonidos para medir el entorno, y giroscopios y acelerómetros para medir el propio movimiento del robot.

MANIPULADORES

La mayoría de los robots actuales se basan en una de las siguientes tres categorías. **Manipuladores**, o brazos robóticos están físicamente anclados en su lugar de trabajo, por ejemplo en una línea de ensamblaje o en la estación espacial internacional. El movimiento de los robots manipuladores normalmente requiere un desplazamiento en cadena de las articulaciones para posicionar a los efectores en cualquier lugar del entorno de trabajo. Los manipuladores son los robots industriales más extendidos con más de un millón de unidades instaladas en el mundo. Algunos manipuladores móviles se utilizan en hospitales para asistir a cirujanos. Pocos fabricantes de coches pueden sobrevivir sin robots manipuladores, incluso algunos robots manipuladores se utilizan para crear ilustraciones y cuadros originales.

[1] En el Capítulo 2 nosotros hablamos sobre los **actuadores,** no sobre efectores. Un actuador es una línea de control que transmite un comando a un efector; el efector es el dispositivo físico en sí mismo.

ROBOTS MÓVILES

La segunda categoría son los **robots móviles**. Los robots móviles se desplazan por su entorno utilizando ruedas, piernas o mecanismos similares. Estos han utilizado para distribuir comidas en hospitales, mover contenedores a los muelles de carga y tareas similares. No hace mucho hemos encontrado un ejemplo de robot móvil: el NAVLAB, **vehículo terrestre sin tripulación** (*unmanned land vehicle*, ULV) capaz de realizar un desplazamiento autónomo para la navegación por autopistas. Otros tipos de robots móviles incluyen los **vehículos aéreos sin tripulación** (*unmanned air vehicles*, UAV), utilizados frecuentemente para tareas de vigilancia, fumigación y operaciones militares; los **vehículos submarinos sin tripulación** (*autonomous underwater vehicles*, AUV) se usan en exploración del fondo marino, paseos planetarios, como el robot Sojourner mostrado en la Figura 25.1(a).

ULV

AAV

AUV

ROBOTS HUMANOIDES

El tercer tipo es un híbrido: un robot móvil equipado con manipuladores. Esto incluye a los **robots humanoides**, cuyo diseño se asemeja al torso humano. La Figura 25.1(b) muestra dos de estos robots, los dos construidos por Honda Corp., en Japón. Los robots híbridos pueden aplicar sus efectores en un campo más amplio que los robots anclados pero las tareas son más complejas porque no tienen la rigidez que el anclaje proporciona.

El campo de la robótica también incluye prótesis artificiales (labios, orejas, ojos…), entornos inteligentes (como una casa equipada con sensores y efectores), y sistemas multicuerpo, cuando la acción robótica se lleva a cabo mediante la cooperación de un enjambre de pequeños robots.

Los robots reales normalmente suelen hacer frente a ambientes que sólo son observables parcialmente, estocásticos, dinámicos y continuos. Algunos de estos ambientes son a la vez secuenciales y multiagente. Los observados parcialmente y los estocásticos son el resultado de interactuar con un entorno grande y complejo. El robot no puede ver

(a) (b)

Figura 25.1 (a) NASA's Sojourner, un robot móvil explorando la superficie de Marte en julio de 1997. (b) Robots Humanoides P3 y Asimo de la casa Honda.

alrededor de las esquinas, y las órdenes de movimiento están sujetas a la incertidumbre generada por el deslizamiento y la fricción de los engranajes. Además el obstinado mundo real rechaza operar más rápido que el tiempo real. En un entorno simulado, es posible utilizar algoritmos sencillos (como el algoritmo **Q-learning** descrito en el Capítulo 21) para aprender en unas pocas horas de proceso realizando millones de pruebas. En un entorno real dichas pruebas tardarían años en realizarse. Además los fallos con un robot real pueden ser peligrosos, cosa que en un sistema simulado no pasa. Los sistemas robóticos prácticos necesitan que se incorpore conocimiento inicial sobre el robot, el ambiente físico y las tareas que el robot realizará para que el robot pueda aprender de manera rápida y segura.

25.2 *Hardware* robótico

Hasta ahora en este libro se ha comentado la arquitectura de agente (sensores, efectores y procesadores) y por tanto nos hemos concentrado en el programa de agentes. El éxito de los robots reales depende como mínimo de que el diseño de los sensores y los efectores sean apropiados para la tarea a realizar.

Sensores

SENSORES PASIVOS

SENSORES ACTIVOS

Los sensores son la interfaz perceptual entre el robot y su entorno. Los **sensores pasivos**, como las cámaras, son observadores verídicos del entorno: capturan las señales que son generadas por otras fuentes en el entorno. Los **sensores activos**, como un sónar, emiten energía dentro del entorno. Se basa en el hecho de que esta energía será reflejada y devuelta al sensor. Los sensores activos suelen proporcionar más información que los pasivos pero a expensas de un mayor consumo y con el peligro de causar interferencias cuando se usan múltiples sensores al mismo tiempo. Sean activos o pasivos los sensores se pueden dividir en tres tipos, dependiendo de si determina distancias a objetos, imágenes del entorno o propiedades del robot en sí mismo.

ESCÁNERES DE RANGO

SENSOR DE SÓNAR

Muchos robots móviles hacen uso de **escáneres de rango**, se trata de sensores que miden la distancia a objetos cercanos. Uno de los más comunes es el **sensor de sónar**, también conocido como transductor ultrasónico. Los sensores de sónar emiten ondas de sonido direccionales, que son reflejadas por objetos, haciendo que parte del sonido vuelva al sensor. El tiempo y la intensidad de la señal de retorno proporcionan información sobre la distancia a los objetos cercanos. Los sónares submarinos son la tecnología elegida por los AUVs. En tierra, los sónares se utilizan principalmente en rangos cercanos para evitar colisiones, debido a su limitada resolución angular. Las alternativas al sónar incluyen el radar (utilizado principalmente en aviación) y el láser. Un escáner de rango basado en láser se muestra en la Figura 25.2.

SENSORES TÁCTILES

Algunos sensores de rango miden distancias cortas o muy cortas. Entre ellos se incluyen los **sensores táctiles** como las antenas[2], interruptores y piel sintética sensible al

[2] *Nota del traductor:* pequeñas antenas sensibles como las de los insectos, del original «whiskers».

Figura 25.2 (a) El escáner de rango SICK LMS basado en Láser, un popular sensor de rango para robots móviles. (b) Lectura de rango obtenida con un sensor montado horizontalmente, proyectada en un mapa de entorno de dos dimensiones.

SISTEMA DE POSICIONAMIENTO GLOBAL (GPS)

tacto. Al otro lado del espectro nos encontramos con **GPS** (sistema de posicionamiento global), el cual mide la distancia a una serie de satélites que emiten señales de pulso. Actualmente hay docenas de satélites en órbita cada uno transmitiendo señales en dos frecuencias diferentes. Los receptores de GPS pueden recuperar la distancia a los satélites analizando los retardos de fase de la señal. Mediante la triangulación de la señal desde múltiples satélites los receptores de GPS pueden determinar la localización absoluta en la tierra con unos pocos metros de error. Los **GPS diferenciales** incluyen un segundo grupo de receptores con conocimiento de la localización, proporcionando una exactitud milimétrica en condiciones ideales. Desafortunadamente, los GPS no funcionan en interiores ni bajo el agua.

GPS DIFERENCIALES

SENSORES DE IMAGEN

La segunda clase importante de sensores son los **sensores de imagen**, las cámaras que nos proporcionan imágenes del entorno y, utilizando las técnicas de visión por computador del Capítulo 24, aplican diversos modelos y extraen características del entorno. La visión estereoscópica es particularmente importante en robótica porque puede capturar información de profundidad; aunque su futuro es incierto ya que se están desarrollando nuevas tecnologías de rango con funcionamiento satisfactorio.

SENSORES PERCEPTORES

DECODIFICADORES DE EJES

La tercera clase importante son los **sensores perceptores**, los cuales informan al robot de su propio estado. Para medir exactamente la configuración de una articulación robótica, los motores están normalmente equipados con **decodificadores de ejes** que cuentan las revoluciones de los motores con pequeños incrementos. En un brazo de robot dichos decodificadores pueden proporcionar medidas exactas sobre cualquier período de tiempo. En un robot móvil los decodificadores informan de las vueltas de las ruedas, estas vueltas pueden ser utilizadas como medidas para la **odometría** (la medida de la distancia recorrida). Desafortunadamente las ruedas tienden a patinar o frenar de manera que la odometría sólo es exacta para desplazamientos muy cortos. Fuerzas exter-

ODOMETRÍA

nas, corrientes marinas para los AUV's y el viento para los UAV incrementan la incertidumbre de su posicionamiento. Los **sensores inerciales**, como los giroscopios, nos pueden ayudar pero no pueden evitar la inevitable acumulación de incertidumbre respecto a la posición del robot.

SENSORES INERCIALES

Otros aspectos importantes del estado del robot son medidos por los **sensores de fuerza y torsión**. Estos son indispensables cuando los robots manejan objetos frágiles u objetos cuya forma y localización son desconocidos. Imagine un robot manipulador de una tonelada manipulando un destornillador con una bombilla. Los sensores de fuerza proporcionan al robot el sentir cómo de fuerte se está agarrando la bombilla y el sensor de torsión cómo de fuerte la está girando. Los buenos sensores pueden medir fuerzas en tres dimensiones y giros en los tres ejes.

SENSORES DE FUERZA Y TORSIÓN

Efectores

Los efectores son el medio por el cual los robots se mueven y cambian la forma de sus cuerpos. Para entender el diseño de los efectores, nos ayudará primero hablar del movimiento y de la forma de manera abstracta, utilizando el concepto de grados de libertad. Nosotros contamos los **grados de libertad** como cada dirección independiente del robot en la cual uno o varios de sus efectores se puedan mover. Por ejemplo un brazo rígido de robot como un robot de tipo AUV que tenga seis grados de libertad, tres de los cuales (x, y, z) lo localizan en el espacio y tres más son su orientación angular conocidas como desvío, balanceo y cabeceo (*yaw, roll* y *pitch*). Estos seis grados definen su **estado cinemático o pose** del robot. El **estado dinámico** del robot incluye una dimensión adicional para evaluar el cambio cinemático de cada dimensión.

GRADOS DE LIBERTAD

ESTADO CINEMÁTICO O POSE

ESTADO DINÁMICO

Para cuerpos no rígidos, hay grados adicionales de libertad dentro del robot. Por ejemplo en un brazo humano el codo tiene un grado de libertad (se puede flexionar en una dirección) y la muñeca tiene tres grados de libertad (se puede mover de arriba abajo, de lado a lado y también puede rotar). Las articulaciones de los robots tienen uno, dos o tres grados de libertad cada una. Seis grados de libertad son los mínimos requeridos para poner un objeto, como una mano, en una posición determinada con una orientación determinada. El brazo mostrado en la Figura 25.3(a) tiene exactamente seis grados de libertad, creado con cinco **articulaciones de revolución** que generan movimiento rotacional y una **articulación prismática** que genera desplazamiento lateral. Se puede verificar que el brazo humano en su totalidad tiene más de seis grados de libertad con un simple experimento: ponga su mano en la mesa, dese cuenta de que aún tiene libertad para rotar su codo sin cambiar la configuración de su mano. Los manipuladores que tienen más grados de libertad que los requeridos para poner un objeto en un lugar determinado son más fáciles de controlar que los robots que sólo tienen el mínimo número de grados de libertad.

ARTICULACIONES DE REVOLUCIÓN

ARTICULACIÓN PRISMÁTICA

Para los robots móviles los grados de libertad no son necesariamente los mismos que el número de elementos actuadores. Considere, por ejemplo, su coche: se puede mover hacia delante y hacia atrás y puede girar, otorgándole dos grados de libertad. En contraste, la cinemática de un coche es tridimensional: en una superficie abierta, uno puede fácilmente mover un coche a cualquier punto (x, y) en cualquier orientación (*véase*

Figura 25.3 (a) El manipulador de Standford, un brazo de robot con cinco articulaciones de revolución (R) y una articulación prismática (P), para un total de seis grados de libertad. (b) Movimiento de un vehículo no holonómico con cuatro ruedas y dirección delantera.

GRADOS EFECTIVOS DE LIBERTAD

GRADOS CONTROLABLES

NO HOLONÓMICO

Figura 25.3(b)). Así pues el coche tiene tres **grados efectivos de libertad** pero sólo dos **grados controlables** de libertad. Se llama robot **no holonómico** a aquel que tenga mas grados de libertad efectivos que controlables y **holonómico** si tienen el mismo número. Los robots holonómicos son más fáciles de controlar (sería más fácil aparcar un coche que se pueda mover de lado además de hacia delante y hacia atrás) pero los robots holonómicos son más complejos mecánicamente. La mayoría de los brazos robóticos son holonómicos al igual que la mayoría de los robots móviles son no holonómicos.

IMPULSIÓN DIFERENCIAL

CONDUCCIÓN SINCRONIZADA

Para los robots móviles, existen un cierto número de mecanismos de movimiento, incluyendo ruedas, orugas y piernas. Los robots de **impulsión diferencial** poseen dos ruedas independientes (u orugas), una a cada lado, como en un tanque militar. Si ambas ruedas se mueven a la misma velocidad el robot se mueve en línea recta. Si se mueven en direcciones opuestas el robot gira sobre sí mismo. Una alternativa es la **conducción sincronizada**, en la cual cada rueda se puede mover y rotar en su propio eje. Esto puede fácilmente llevar al caos, si no se limita que todas las ruedas siempre apunten a la misma dirección y se muevan a la misma velocidad. Tanto la impulsión diferencial como la conducción sincronizada son no holonómicas. Algunos robots más caros utilizan manejadores holonómicos, los cuales utilizan normalmente tres o más ruedas que pueden ser movidas y orientadas independientemente.

DINÁMICAMENTE ESTABLE

ESTÁTICAMENTE ESTABLE

Las piernas robóticas, contrariamente a las ruedas, pueden manejarse por terreno abrupto. No obstante las piernas son notoriamente lentas en superficies planas además de ser mecánicamente difíciles de construir. Los investigadores en robótica han tratado de diseñar robots desde una hasta múltiples piernas. Los robots con piernas pueden caminar, correr, saltar (como se puede ver en el robot de la Figura 25.4(a)). Este robot es **dinámicamente estable**, es decir, puede permanecer vertical mientras camina. Un robot que pueda permanecer vertical sin mover sus piernas se llama **estáticamente estable**. Un robot es estáticamente estable si su centro de gravedad está encima del polígono que une sus piernas.

Otros tipos de robots móviles utilizan una gran cantidad de mecanismos diferentes para desplazarse. Los vehículos de remolque normalmente utilizan propulsores o turbinas. Los dirigibles robóticos se basan en efectos termodinámicos que los mantienen a

Figura 25.4 (a) Uno de los robots con piernas de Marc Raibert. (b) Robot AIBO de Sony.

flote. Los vehículos autónomos submarinos normalmente utilizan propulsores similares a los usados en submarinos.

MOTOR ELÉCTRICO

ACTUACIÓN NEUMÁTICA

ACTUACIÓN HIDRÁULICA

Por sí solos los sensores y efectores no hacen un robot. Un robot completo también necesita una fuente de energía para hacer funcionar sus efectores. El **motor eléctrico** es el mecanismo más popular para la manipulación y la locomoción pero la **actuación neumática**, utilizando gas comprimido y la **actuación hidráulica**, utilizando fluidos presurizados, también tienen sus nichos de aplicación. La mayoría de los robots también tienen medios para la comunicación digital como una red inalámbrica. Por último tiene que haber un cuerpo para ensamblar las diferentes piezas del robot.

25.3 Percepción robótica

La percepción es el proceso por el cual los sensores del robot elaboran un mapa del entorno utilizando una determinada representación interna. La percepción es complicada porque en general los sensores tienen ruido y el entorno es parcialmente observable, impredecible y dinámico. En general una buena representación interna ha de tener tres propiedades: tiene que contener la información suficiente para que el robot tome las decisiones apropiadas, tiene que estar estructurada para que se pueda actualizar eficientemente y tiene que ser natural, es decir, las variables internas han de corresponder con estados en el mundo real.

En el Capítulo 15, nosotros decimos que los filtros de Kalman, HMMs, y las redes Bayesianas dinámicas pueden representar las transiciones y los modelos de sensores a partir de entornos observables, y nosotros describimos los algoritmos para actualizar los **estados de creencia** (la distribución de la probabilidad a posteriori en el entorno de las variables de estado). Algunos modelos dinámicos de redes de Bayes para este proceso se muestran en el Capítulo 15. Para los problemas de robótica normalmente se incluyen las acciones ya realizas por el robot como variables observadas (*véase* Figura 17.9). La Figura 25.5 muestra la notación utilizada en este capítulo: $\mathbf{X}_t$ es el estado del entorno (incluyendo el robot) en el instante t, $\mathbf{Z}_t$ es la observación recibida en el instante t, y A_t es la acción realizada después de recibir la observación.

La tarea de **filtrar**, o actualizar la creencia de estados, es esencialmente la misma que se ha descrito en el Capítulo 15. Hay que computar la nueva creencia del estado, $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t})$, desde el estado actual $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ y la nueva observación $\mathbf{z}_{t+1}$. Las principales diferencias son que (1) estamos condicionados explícitamente por las acciones, así como por las observaciones, y (2) ahora nos enfrentamos a variables continuas y no discretas. Así, nosotros modificamos el filtrado recursivo de la Ecuación 15.3 para utilizar integración en vez de suma:

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t}) = \alpha \mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)\, P(\mathbf{x}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})\, d\mathbf{x}_t. \qquad (25.1)$$

La ecuación establece que el valor posterior a las variables de estado $\mathbf{X}$ en el instante de tiempo $t + 1$ es calculado recursivamente a partir del estado correspondiente estimado un instante de tiempo antes. Este cálculo incluye la acción previa a_t y la actual medida $\mathbf{z}_{t+1}$. Por ejemplo, si nuestra meta es desarrollar un robot que juegue al fútbol, $\mathbf{X}_{t+1}$ puede ser la localización de la pelota con respecto al robot. El posterior $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ es la probabilidad de distribución en todos los estados que capturan qué sabemos de las medidas realizadas por el sensor. La Ecuación (25.1) nos dice cómo obtener recursivamente esta localización, incrementando las medias de los sensores (por ejemplo, con imágenes) y los comandos de movimiento del robot. La probabilidad $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)$ se llama el **modelo transicional** o **modelo de movimiento**, y $\mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1})$ es el **modelo del sensor.**

MODELO DE MOVIMIENTO

Figura 25.5 La percepción de un robot puede ser vista como una inferencia temporal de secuencias de acciones y medidas, como se ilustra en esta red dinámica de Bayes.

LOCALIZACIÓN

Localización

La localización es un ejemplo genérico de percepción robótica. Es el problema de determinar dónde están las cosas. La localización es uno de los problemas más relevantes de la percepción robótica debido a que el conocimiento sobre dónde están las cosas es el núcleo principal para una satisfactoria interacción física. Por ejemplo, los manipuladores del robot deben saber la localización de los objetos que han de manipular. Los robots navegadores deben saber dónde están para poder encontrar las localizaciones requeridas.

PROBLEMA DE SEGUIMIENTO, TRACKING

LOCALIZACIÓN GLOBAL

PROBLEMA DEL SECUESTRO

El problema de la localización viene dado en tres formas con dificultad incremental. Si la posición inicial del objeto es conocida, entonces la localización es un **problema de seguimiento** (*tracking*). Los problemas de seguimiento se caracterizan por una incertidumbre limitada. Más difícil es el problema de la **localización global**, en el cual la posición inicial del objeto es desconocida. La localización global se convierte en el problema de seguimiento una vez que el objeto de interés ha sido localizado, este también incluye fases en las que el robot tiene que manejar incertidumbre. Para terminar nosotros podemos ser malos con nuestro robot y «secuestrar» el objeto que está intentando localizar. La localización en las condiciones anteriores es conocida como **el problema del secuestro**. El secuestro se utiliza normalmente para probar lo robusta que es una técnica de localización bajo condiciones extremas.

Para simplificar, asumamos que el robot se mueve lentamente en un plano y que se le da un mapa exacto del entorno. La posición de dicho móvil se define por sus dos coordenadas cartesianas con los valores x, y, y la orientación θ, como se ilustra en la Figura 25.6(a). (Obsérvese que se han excluido las correspondientes velocidades, por lo que es un modelo cinemático en vez de uno dinámico.) Si ordenados dichos valores en un vector, entonces cualquier estado particular está dado por $\mathbf{X}_t = (x_t, y_t, \theta_t)^\top$.

En la aproximación cinemática, cada acción consiste es una especificación «instantánea» de dos velocidades, una velocidad transaccional v_t y una velocidad rotacional w_t. Para pequeños intervalos Δ_t, un modelo determinístico de movimiento de dicho robot viene dado por:

$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t, w_t}_{a_t}) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta t \cos \theta_t \\ v_t \Delta t \operatorname{sen} \theta_t \\ w_t \Delta t \end{pmatrix}$$

La notación $\hat{\mathbf{X}}$ se refiere a un estado determinístico de predicción. Por supuesto, los robots físicos son algo impredecibles. Éste está normalmente modelado usando una distribución gaussiana con media $f(\mathbf{X}_t, v_t, w_t)$ y una covarianza Σ_x. (*Véase* el Apéndice A para la definición matemática).

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, v_t, w_t) = N(\hat{\mathbf{X}}_{t+1}, \Sigma_x)$$

MARCADORES

Después, nosotros necesitamos un modelo de sensor. Se considerarán dos clases de modelos de sensor. La primera asume que el sensor detecta características estables y reconocibles del entorno, llamadas **marcadores**. Para cada marcador se informa de su rango y capacidad. Supóngase que el estado del robot es $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$ y se detecta un mar-

Figura 25.6 (a) Un modelo simplificado de cinemática para un robot móvil. El robot se muestra como un círculo con una marca mostrando la dirección a la que mira el robot. Las posiciones y las orientaciones en t y $t + 1$ se muestran, con las actualizaciones dadas por $v_t\Delta_t$ y $w_t\Delta_t$ respectivamente. También se muestra un marcador en (x_i, y_i) observado en el instante t. (b) El modelo del escaneador de rango. Dos posibles poses del robot se muestran para una medida de rango dada (z_1, z_2, z_3, z_4). Es mucho más probable que la pose de la izquierda genere el modelo del escaneador de rango que la pose de la derecha.

cador cuya localización es conocida como $(x_t, y_t)^\top$. Sin ruido, el rango y la capacidad pueden calcularse por simple geometría (*véase* la Figura 25.6(a)). La predicción exacta del rango e intensidad observadas sería:

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \dfrac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}$$

Otra vez, el ruido distorsiona nuestras medidas. Para simplificar, podemos suponer un ruido gaussiano con covarianza Σ_z

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = N(\hat{\mathbf{z}}_t, \Sigma_z)$$

Un modelo diferente de sensor es a menudo más apropiado para los sensores de rango de la clase mostrada en la Figura 25.2. Dichos sensores producen un vector de valores $\mathbf{z}_t = (z_1, ..., z_M)^\top$, con intensidad fijada por el robot, dada una posición $\mathbf{x}_t$, siendo $\hat{z}_j$ el rango de la j-ésima dirección desde $\mathbf{x}_t$ al obstáculo más cercano. Como antes, se utilizará ruido gaussiano. Normalmente asumimos que los errores para las diferentes direcciones de medida son independientes y tienen distribución idéntica, con lo que tenemos:

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \alpha \prod_{j=1}^{M} e^{-(z_j - \hat{z}_j)/2\sigma^2}$$

La Figura 25.6(b) muestra un ejemplo de un escáner de cuatro haces y dos posibles posiciones del robot, una de la cual es supuestamente producida por la observación del escáner y la otra no. Comparando el modelo del escáner de rango con el modelo del

marcador, podemos ver que el modelo del escáner de rango tiene la ventaja de que no es necesario *identificar* el marcador antes de que el escáner pueda ser interpretado; de hecho, en la Figura 25.6(b), el robot hace frente a una pared sin rangos distintivos. Por otra parte, si hay un marcador distintivo éste nos puede ayudar a localizarnos inmediatamente.

En el Capítulo 15 se describe el filtro de Kalman, que representa la creencia de un estado como una única función multivariada de Gauss, y el filtro de partículas, que representa el estado de creencia de una colección de partículas que corresponden a estados. Los algoritmos modernos de localización usan una o dos representaciones de la creencia del robot $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$.

LOCALIZACIÓN DE MONTE-CARLO

La localización utilizando filtrado de partículas se llama **localización de Monte-Carlo** o MCL. El algoritmo MCL es esencialmente idéntico al algoritmo de filtrado de partículas mostrado en la Figura 15.15. Todo lo que necesitamos es proporcionar el modelo apropiado de movimiento y modelo del sensor. La Figura 25.7 nos muestra una versión utilizando el modelo del escáner de rango. El funcionamiento del algoritmo se ilustra en la Figura 25.8 donde el robot localiza su posición dentro de un edificio de oficinas. En la primera imagen las partículas están uniformemente distribuidas basadas en el conocimiento a priori, indicando la incertidumbre global sobre la posición del robot. En la segunda imagen, el primer conjunto de medidas hace que las partículas se agrupen en áreas de alta creencia a posteriori. En la tercera hay suficientes medidas para que todas las partículas estén en una posición determinada.

El filtro de Kalman es otra manera de localizar. Un filtro de Kalman representa la creencia a posterior $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ con una gaussiana. La media de esta gaussiana se denota por μ_t y su covarianza Σ_t. El problema principal con la creencia gaussiana es que

```
función LOCALIZACIÓN-MONTE-CARLO(a, z, N, modelo, mapa) devuelve un conjunto de muestras
  entradas: a, la orden previa de movimiento del robot
            z, un escáner de rango con M lecturas, de z_1 a z_M
            N, el número de muestras a ser mantenidas
            modelo: un modelo probabilista del entorno con posición inicial P(X_0),
                modelo de movimiento P(X_1 | X_0, A_0) y rango del sensor con ruido P(Z | Ẑ).
            mapa: un mapa en dos dimensiones del entorno
  estática: S, un vector de muestras de tamaño N, inicialmente generado por P(X_0)
  variables locales: W, un vector de pesos de tamaño N

  para i = 1 hasta N hacer
      S[i] ← muestra de P(X_1 | X_0 = S[i], A_0 = a)
      W[i] ← 1
      para j = 1 hasta M hacer
         ẑ ← RANGO-EXACTO(j, S[i], mapa)
         W[i] ← W[i] · P(Z = z_j | Ẑ = ẑ)
  S ← MUESTRA-PONDERADA-CON-REEMPLAZAMIENTO(N, S, W)
  devolver S
```

Figura 25.7 Algoritmo de localización de Monte-Carlo usando un modelo de escáner de rango con ruido independiente.

(a)

(b)

(c)

Figura 25.8 Algoritmo de localización Monte-Carlo, un filtro de partículas para localización móvil de robots. Arriba: inicialmente incertidumbre global. Centro: incertidumbre después de navegar por el pasillo. Bajo: incertidumbre después de entrar en una oficina distintiva.

Figura 25.9 Ilustración unidimensional de un modelo lineal de movimiento: (a) Función f, y la proyección de una media μ_t y un intervalo de covarianza (basado en Σ_t) dentro del tiempo $t+1$. (b) La versión lineal es la tangente de f en μ_t. La proyección de media μ_t es correcta. No obstante, la covarianza proyectada Σ_{t+1} difiere de Σ_{t+1}.

sólo se garantiza para modelos lineales f y modelos de medidas lineales h. Para modelos f y h no lineales el resultado de actualizar el filtro normalmente no es una gaussiana. Así, los algoritmos que utilizan el filtro de Kalman para localización **linealizan** el movimiento y los modelos de los sensores. La linealización es una aproximación local de una función no lineal por una función lineal. La Figura 25.9 ilustra el concepto de la linealización para un modelo (unidimensional) de movimiento del robot. En al izquierda se representa un modelo no lineal de movimiento $f(\mathbf{x}_t, a_t)$ (el control de a_t se omite en este grafo ya que no es representativo para la linealización). En la derecha, esta función se aproxima por una función lineal $\tilde{f}(\mathbf{x}_t, a_t)$. Esta función lineal es tangente a f en el punto μ_t, la media de nuestro estado estimado en el instante de tiempo t. Dicha linealización se llama **Expansión de Taylor** (de primer grado). Un filtro de Kalman que linealiza f y h vía expansiones de Taylor se llama **filtro de Kalman extendido** o (EKF). La Figura 25.10 muestra una secuencia de estimaciones de un robot sobre el que funciona el algoritmo de localización basado en un filtro de Kalman. Cuando se mueve el robot, la incertidumbre de su localización se incrementa, como se muestra en el tama-

LINEALIZACIÓN

EXPANSIÓN DE TAYLOR

Figura 25.10 Ejemplo de localización utilizando el filtro extendido de Kalman. Los movimientos del robot en una línea recta. Conforme progresa su incertidumbre se incrementa gradualmente, como se ilustra por las elipses de error. Cuando se observa un marcador con una posición conocida la incertidumbre se reduce.

ño de sus elipses de error. El error disminuye conforme va detectando el marcador. El error finalmente se incrementa otra vez cuando el robot pierde dicho marcador. Los algoritmos EKF funcionan correctamente si los marcadores se identifican fácilmente. En otro caso, la distribución a posteriori puede ser multimodal como en la Figura 25.8(b). El problema de la necesidad de conocimiento de la identificación de marcadores es un ejemplo del problema de **asociación de datos** discutido al final del Capítulo 15.

Generación de mapas

Hasta ahora hemos discutido el problema de la localización de un único objeto. En robótica habitualmente se intenta localizar muchos objetos. El ejemplo clásico de dicho problema es la construcción de mapas mediante robots. Imagine un robot al cual no se le da un mapa del entorno. El robot ha de construirse su propio mapa. Claramente los humanos hemos desarrollado habilidades para realizar mapas, desde lugares pequeños hasta del mundo entero. Por tanto un problema a abordar por la robótica es diseñar algoritmos para que los robots puedan hacer lo mismo.

LOCALIZACIÓN Y MAPEADO SIMULTÁNEO

En la literatura, a la generación de mapas mediante robots muchas veces se le denomina **localización y mapeado simultáneo**, abreviado como **SLAM**. No sólo el robot ha de ser capaz de construir el mapa, lo debe hacer sin ningún conocimiento sobre dónde está. SLAM es uno de los problemas principales de la robótica. Consideraremos la versión en la cual el entorno es fijo. Esto es suficientemente complicado; aun así es mucho más complejo cuando el entorno puede cambiar mientras que el robot se mueve.

Desde una perspectiva estadística, la generación de mapas es un problema de inferencia Bayesiano, como la localización. Si denotamos el mapa por M y la posición del robot en el tiempo t por $\mathbf{X}_t$ como anteriormente, podemos rescribir la Ecuación (25.1) para incluir el mapa a posteriori:

$$\mathbf{P}(\mathbf{X}_{t+1}, M \mid \mathbf{z}_{1:t+1}, a_{1:t}) = \alpha \mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}, M) \int \mathbf{P}(\mathbf{X}_{t+1}, \mid \mathbf{x}_t, a_t)\, P(\mathbf{x}_t, M \mid \mathbf{z}_{1:t}, a_{1:t-1})\, d\mathbf{x}_t$$

Esta ecuación nos trae buenas noticias: la distribución condicional necesita incorporar acciones y medidas que son esencialmente las mismas que en el problema de la localización del robot. La advertencia principal es que el nuevo espacio de estados (el espacio de todas las poses del robot y todos los mapas) tiene muchas dimensiones. Imagine que quiere representar un edificio entero de manera fotográfica. Esto posiblemente requeriría cientos de millones de números. Cada número sería una variable aleatoria y contribuiría a la enorme dimensión del espacio de estados. Lo que hace este problema aún más difícil es el hecho de que el robot no tiene por qué saber cómo de grande es el entorno. Así la dimensión de M tiene que ser ajustada dinámicamente durante el mapeado.

Probablemente el método más utilizado para el problema SLAM es el EKF. Este usualmente se combina con un modelo de detección de marcadores y requiere que todos los marcadores sean distinguibles. En la sección anterior la estimación a posteriori se representó utilizando una Gaussiana con una media μ_t y covarianza Σ_t. En la aproximación del EKF al problema SLAM, la estimación posterior es otra vez gaussiana pero ahora la media μ_t es un vector mucho más grande. Comprende no sólo la pose del ro-

bot sino también la localización de todas las características (o marcadores) en el mapa. Si hay n marcadores, este vector sería de dimensión $2n + 3$ (mantiene dos valores para especificar la localización y tres para la posición del robot). Consecuentemente, la matriz Σ_t es de dimensiones $(2n + 3)$ por $(2n + 3)$. Posee la siguiente estructura:

$$\Sigma_t = \begin{pmatrix} \Sigma_{XX} & \Sigma_{XM} \\ \Sigma_{XM}^\top & \Sigma_{MM} \end{pmatrix} \tag{25.2}$$

Aquí Σ_{XX} es la covarianza de la pose del robot, la cual también la encontramos en el contexto de la localización. Σ_{XM} es una matriz de tamaño $3 \times 2n$ que expresa la correlación entre características del mapa y las coordenadas del robot. Para terminar, Σ_{MM} es la matriz de tamaño $2n \times 2n$ que especifica la covarianza característica en el mapa, incluyendo todos los pares de correlaciones. La memoria requerida para los algoritmos EKF es cuadrática en n, el número de características del mapa y el tiempo de actualización es también cuadrático respecto a n.

Antes de introducirnos en detalles matemáticos, vamos a estudiar la solución del algoritmo EKF gráficamente. La Figura 25.11 muestra un robot en un entorno con ocho marcadores, ordenados en dos filas de cuatro marcadores cada uno. Inicialmente, el robot no sabe dónde están localizados. Supondremos que cada marcador será de un color distinto, y el robot será capaz de detectar cuál es cuál. El robot empieza en una localización bien definida hacia la derecha, pero gradualmente pierde certeza de dónde está. Esto se indica por las elipses de error de la Figura 25.11(a), las cuales aumentan cuando el robot se mueve hacia delante. Mientras se mueve el robot, detecta el rango y la distancia de los marcadores cercanos, y esas observaciones hacen que el robot estime la posición de los mismos. Naturalmente, la incertidumbre de dicha estimación está unida a la incertidumbre de localización del robot. La Figura 25.11(b) y (c) ilustra la creencia del robot mientras se mueve cada vez más lejos en su entorno.

Un detalle importante de esas estimaciones (el cual no es obvio a partir de nuestra representación gráfica) es el hecho de que se mantiene una *única* Gaussiana para todas las estimaciones. Las elipses de error en la Figura 25.11 son proyecciones de esa Gaussiana en subespacios del robot y las coordenadas de los marcadores. Esta Gaussiana multivariada mantiene posteriormente correlaciones entre todas las estimaciones. Esta observación es importante para entender qué ocurre en la Figura 25.11(d). Aquí el robot observa un marcador previamente detectado. Como resultado, su incertidumbre disminuye drásticamente al igual que la incertidumbre del conjunto de marcadores. Esto es una consecuencia del hecho de que las estimaciones del robot y los marcadores están íntimamente correlacionadas con la Gaussiana posterior. Encontrar conocimiento sobre una variable (aquí la pose del robot) automáticamente reduce la incertidumbre en todas las otras.

El algoritmo EKF para generación de mapas se asemeja al algoritmo EKF de localización de la sección previa. La diferencia principal estriba en las variables de los marcadores añadidas a posteriori. El modelo de movimiento para los marcadores es trivial: no se mueven. La función f, por lo tanto, es la función identidad para esas variables. La función de medida es esencial al igual que en el caso anterior. La única diferencia es que la función Jacobiana H_t en la ecuación de actualización de EKF se obtiene no sólo con respecto a la posición del robot sino también respecto a la localización de los marcado-

Figura 25.11 EKF aplicado al problema de generación de mapas robóticos. El camino del robot se muestra en línea discontinua y su estimación de posición en elipses sombreadas. Ocho marcadores distinguibles de posición desconocida se muestran como puntos pequeños, y sus estimadores de localización se muestran como elipses blancas. En (a)-(c) la incertidumbre del robot se va incrementando así como la incertidumbre de encontrarse un marcador. En (d) el robot detecta el primer marcador de nuevo y la incertidumbre de *todos* los marcadores disminuye gracias a que las estimaciones están correlacionadas.

res que son observados en el tiempo t. Las ecuaciones EKF resultantes son incluso más horrorosas que las indicadas anteriormente por lo cual se omitirán aquí.

No obstante, hay otras dificultades que hemos ignorado: el hecho de que el tamaño del mapa M sea desconocido. Por lo tanto el número de elementos en la estimación final μ_t y Σ_t son desconocidos. Se tiene que determinar dinámicamente si el robot descu-

bre nuevos marcadores. La solución a este problema es directa; si el robot descubre un nuevo marcador, añade un nuevo elemento a la información a posteriori. Inicializando la varianza de este nuevo elemento a un valor muy elevado, el resultado a posterior es el mismo que si el robot hubiera conocido la existencia del marcador con antelación.

Otros tipos de percepción

No toda la percepción robótica versa sobre la localización y la generación de mapas. Los robots también reciben la temperatura, olores, señales acústicas, etc. Muchas de estas cantidades se pueden estimar probabilísticamente, como la localización y la generación de mapas. Todo lo que se necesita para dichos estimadores son las distribuciones de probabilidad condicionada que describan la relación entre las medidas y los estados de las variables.

No obstante, no todos los sistemas de percepción robótica se basan en representaciones probabilísticas. En efecto, mientras que el estado interno de todos nuestros ejemplos tienen una interpretación física clara, esto no tiene por qué ser el caso. Por ejemplo, imagine un robot con piernas que intenta levantar una pierna por encima de un obstáculo. Suponga que el robot usa una regla por la cual inicialmente mueve la pierna a una altura baja y luego la sube cada vez más alto si la altura anterior resulta en una colisión con este obstáculo. ¿Diríamos que la altura de la pierna es una representación de alguna cantidad física en el mundo? Puede ser, pero se relaciona con la altura y lo escarpado del obstáculo. No obstante podemos pensar en la altura de la pierna como una variable auxiliar del controlador del robot, desprovisto de significado físico directo. Dicha representación es común en robótica y para algunos problemas funciona correctamente.

La robótica claramente tiende a representaciones con semánticas bien definidas. Las técnicas probabilísticas, en muchos casos, superan a otras aproximaciones de problemas perceptuales como localización y generación de mapas. No obstante, las técnicas estadísticas algunas veces son demasiado incómodas y soluciones más sencillas pueden ser efectivas en la práctica. El mejor profesor para decidir qué aproximación tomar, es la propia experiencia trabajando con robots reales.

25.4 Planear el movimiento

MOVIMIENTO PUNTO A PUNTO

MOVIMIENTO OBEDIENTE

En robótica, las decisiones últimamente incluyen el movimiento de efectores. El **movimiento punto a punto** consiste en situar al robot o a sus efectores en una posición determinada. Un reto aún mayor es el problema del **movimiento obediente**, en el cual un robot se desplaza mientras está en contacto físico con un obstáculo. Un ejemplo de movimiento obediente es un robot que manipula un destornillador con una bombilla o un robot que pone una caja encima de una mesa.

Empezaremos encontrando una buena representación en la cual los problemas de planificación puedan ser descritos y solucionados. Resulta que el **espacio de configuración** (el espacio de los estados del robot definido por la localización, orientación y ángulos

PLANIFICACIÓN DE TRAYECTORIAS

de las articulaciones) es un lugar mejor para trabajar que el espacio original en tres dimensiones. El **problema de planificación de trayectorias** consiste en encontrar un camino de una configuración a otra configuración del espacio. Hemos encontrado varias versiones del problema de planificación en este libro; en robótica, la característica principal de la planificación es que requiere espacios *continuos*. La literatura en planificación distingue un conjunto de técnicas diferentes específicamente creadas para encontrar caminos en un espacio continuo de dimensión alta. Las aproximaciones más comunes se conocen como **descomposición en celdas** y **esqueletización**. Cada una reduce el problema continuo de planificación en la búsqueda sobre un grafo discreto mediante la identificación de algunos estados canónicos y caminos entre el espacio libre. En esta sección se asume que el movimiento es determinístico y la localización del robot es exacta. Las secciones siguientes relajarán esas condiciones.

Espacio de configuración

El primer paso hacia la solución del problema es encontrar una representación apropiada para el problema. Empezaremos con una simple representación para un problema sencillo. Considere el brazo de robot mostrado en la Figura 25.12(a). Tiene dos articulaciones que se mueven independientemente. Moviendo las articulaciones altera las coordenadas (x, y) del codo y de la pinza. (El brazo no puede moverse en la dirección de z.) Esto sugiere que la configuración puede ser descrita por una coordinación cuatro-dimensional: (x_e, y_e) para la localización del codo relativa al entorno y (x_g, y_g) para la localización de la pinza. Obviamente, estas cuatro coordenadas caracterizan el estado del robot. Ellas constituyen lo que se conoce como representación del **área de trabajo**, en la que las coordenadas del robot se especifican con el mismo sistema de coordenadas que los objetos que busca para manipular (o evitar). La representación del área de trabajo está bien adaptada para la comprobación de colisiones, especialmente si el robot y todos los objetos se pueden representar por modelos poligonales simples.

ÁREA DE TRABAJO

El problema con la representación del área de trabajo es que no todas las coordenadas de dicha área son realmente alcanzables, incluso con la ausencia de obstáculos. Eso es así debido a las **restricciones de unión** en el espacio para alcanzar las coordenadas del área de trabajo. Por ejemplo, la posición del codo (x_e, y_e) y de la pinza (x_g, y_g) distan siempre una distancia fija, porque están unidas por un brazo rígido. Un planificador de movimiento de un robot definido sobre un área de trabajo con coordenadas se enfrenta al problema de generar soluciones que soporten dichas limitaciones. Esto es particularmente difícil porque el espacio de estados es continuo y las limitaciones no son lineales.

RESTRICCIONES DE UNIÓN

ESPACIO DE CONFIGURACIÓN

Resulta más fácil planificar con una representación del **espacio de configuración.** En vez de representar el estado del robot por las coordenadas cartesianas de sus elementos, representamos su estado por la configuración de sus articulaciones. Nuestro robot de ejemplo tiene dos articulaciones. Por lo tanto, podemos representar su estado con dos ángulos α_s y α_e para la articulación del hombro y del codo respectivamente. En ausencia de obstáculos, un robot puede situarse en cualquier posición del área de trabajo. En particular, cuando se planifica un determinado camino se puede simplemente

conectar la configuración actual y la de la meta por una línea recta. En el siguiente camino, el robot debería cambiar sus articulaciones a velocidad constante, hasta que se alcance la meta.

Desafortunadamente, el espacio de configuración tiene sus problemas. La tarea de un robot normalmente se expresa mediante coordenadas del área del trabajo, no en coordenadas del espacio de configuración. Por ejemplo podremos querer que un robot mueva su efector final a una cierta coordenada del área de trabajo, posiblemente con una especificación de su orientación. Esto lleva a la pregunta de cómo convertir las coordenadas del área de trabajo en las del espacio de trabajo. En general la *inversa* de este problema, transformar configuraciones del espacio de coordenadas en coordenadas del área de trabajo es simple: incluye una serie de transformaciones obvias del espacio de coordenadas. Estas transformaciones son lineales para las articulaciones prismáticas y trigonométricas para articulaciones de revolución. Esta cadena de transformaciones de coordenadas es conocida como **cinemática**, un término que encontramos en la discusión sobre los robots móviles.

El problema inverso de calcular la configuración del robot cuya localización de los efectores se especifica en coordenadas del área de trabajo es conocido como **cinemática inversa**. Calcular la cinemática inversa es normalmente difícil sobre todo para robots con muchos grados de libertad. En particular la solución es pocas veces única. Por ejemplo nuestro brazo de robot tiene dos configuraciones distintas con las cuales las pinzas llegan a las mismas coordenadas del área de trabajo como se muestra en la figura.

CINEMÁTICA INVERSA

En general, este brazo de robot de dos articulaciones está entre cero y dos soluciones de cinemática inversa para cada conjunto de coordenadas del área de trabajo. La mayoría de los robots industriales tienen infinitas soluciones. Para ver cómo puede ser esto posible, simplemente imagine que añadimos una tercera articulación de revolución a nuestro robot de ejemplo, una cuyo eje rotacional sea paralelo a una de las articulaciones ya existentes. En dicho caso podemos alcanzar la localización (pero no la orientación) de la pinza fija y rotar sus articulaciones internas, para la mayoría de las configuraciones del robot. Con unas pocas articulaciones más (¿cuántas?) podemos obtener el mismo efecto mientras se mantiene la orientación constante. Hemos visto ya un ejemplo de esto en el «experimento» de colocar su mano en el escritorio y mover su codo. La restricción cinemática de la posición de la mano es insuficiente para determinar la configuración del codo. En otras palabras, la cinemática inversa del ensamblaje hombro-brazo posee un número infinito de soluciones.

El segundo problema con las representaciones del espacio de configuración se encuentra en los obstáculos que pueden existir en el área de trabajo del robot. Nuestro ejemplo en la Figura 25.12(a) muestra varios de estos obstáculos, incluyendo un obstáculo colgante que se introduce en el interior del área de trabajo del robot. En el área de trabajo, estos obstáculos tienen formas geométricas simples. Pero, ¿qué aspecto tienen en el espacio de configuración?

La Figura 25.12(b) muestra el espacio de configuración para nuestro robot de ejemplo, bajo la configuración específica de obstáculos mostrada en la Figura 25.12(a). El espacio de configuración puede ser descompuesto en dos subespacios: el espacio de todas las configuraciones que el robot puede alcanzar, comúnmente llamado **espacio libre**, y el espacio de las configuraciones inalcanzables, llamado **espacio ocupado**. El área

ESPACIO LIBRE

ESPACIO OCUPADO

Figura 25.12 (a) Representación del área de trabajo de un brazo robótico con dos grados de libertad. El área de trabajo es una caja con un obstáculo plano colgando del techo. (b) El espacio de configuración del mismo robot. Sólo las regiones blancas en el espacio son configuraciones libres de colisión. El punto en este diagrama se corresponde con la configuración del robot mostrada a la izquierda.

blanca en la Figura 25.12(b) se corresponde al espacio libre. Todas las demás regiones se corresponden con espacio ocupado. El sombreado diferente del espacio ocupado se corresponde con los distintos objetos en el área de trabajo del robot; la región negra que rodea el espacio libre por entero son las configuraciones en las que el robot colisiona consigo mismo. Es fácil ver que valores extremos para los ángulos del hombro o el codo pueden causar un choque de este tipo. Las dos regiones ovales a ambos lados del robot corresponden a la mesa donde el robot está montado. Similarmente, la tercera región oval corresponde a la pared izquierda. Por último, el objeto más interesante en el espacio de configuración es un simple obstáculo vertical entorpeciendo el área de trabajo del robot. Este objeto en el espacio de configuración tiene una forma extraña: es altamente no lineal e incluso en algunos lugares cóncava. Con una pequeña cantidad de imaginación el lector reconocerá la forma de la pinza en el extremo superior izquierdo. Recomendamos al lector parar por un momento y estudiar este importante diagrama. ¡La forma de este obstáculo no es del todo obvia! El punto en el interior de la Figura 25.12(b) marca la configuración del robot, como muestra la Figura 25.12(a). La Figura 25.13 representa tres configuraciones adicionales, tanto en el espacio de configuración como en el área de trabajo. En la configuración «conf-1», la pinza circunda el obstáculo vertical.

En general, incluso si el área de trabajo del robot se representa por polígonos planos, la forma del espacio libre puede ser muy complicada. En la práctica, en consecuencia, se suele *investigar* un espacio de configuración en lugar de construirlo explícitamente. Un planificador puede generar una configuración y entonces comprobar si está en es-

Figura 25.13 Tres configuraciones del robot, mostradas en el área de trabajo y el espacio de configuración.

pacio libre aplicando la cinemática del robot y entonces comprobar si se producen colisiones en las coordenadas del área de trabajo.

Métodos de descomposición en celdas

DESCOMPOSICIÓN EN CELDAS

Nuestra primera aproximación a la planificación de rutas utiliza **descomposición en celdas**, esto es, descompone el espacio libre en un número finito de regiones continuas, llamadas celdas. Estas regiones tienen la importante propiedad de que el problema de planificación de rutas en una región simple puede ser resuelta de forma simple (por ejemplo, moviéndose en línea recta). El problema de planificación de rutas se transforma entonces en un problema de búsqueda en un grafo discreto, muy parecido a los problemas de búsqueda mostrados en el Capítulo 3.

La descomposición en celdas más simple consiste en una rejilla regular. La Figura 25.14(a) muestra una descomposición en una rejilla cuadriculada del espacio y una ruta solución que es óptima para este tamaño de rejilla. Hemos utilizado también un sombreado en escala de grises para indicar el *valor* de cada celda de la rejilla de espacio libre, por ejemplo, el coste del camino más corto desde la celda hasta el objetivo. (Estos valores pueden ser calculados de una forma determinista mediante el algoritmo de VALUE-ITERATION dado en la Figura 17.4.) La Figura 25.14(b) muestra la trayectoria correspondiente en el área de trabajo del brazo.

Esta descomposición tiene la ventaja de que es extremadamente fácil de implementar, pero sufre dos limitaciones. Primero, sólo podemos trabajar con espacios de configuración de poca dimensionalidad, ya que el número de celdas de la rejilla crece

(a) (b)

Figura 25.14 (a) Función de valor y ruta encontrada para una aproximación de rejilla discreta de celdas del espacio de configuración. (b) La misma ruta visualizada en coordenadas del área de trabajo. Observe cómo el robot dobla su codo para evitar la colisión con el obstáculo vertical.

exponencialmente con d, el número de dimensiones. Segundo, existe el problema de qué hacer con las células «mixtas», esto es, que no se encuentran por entero ni en espacio libre ni en espacio ocupado. Una solución que incluya una celda de este tipo puede que no sea una solución real, porque puede no haber forma de cruzar la celda en la dirección deseada mediante una línea recta. Esto puede hacer el planificador de rutas *imperfecto*. Por otra parte, si hacemos que sólo las celdas completamente libres puedan ser usadas, el planificador sería *incompleto*, ya que se puede dar el caso de que las únicas rutas al objetivo deban ir a través de celdas mixtas, especialmente si el tamaño de la celda es comparable a los pasillos o los espacios libres en el espacio.

Hay dos formas de modificar el método de descomposición en celdas para evitar estos problemas. Lo primero es permitir una *subdivisión adicional* de las celdas mixtas (quizás usando celdas con un tamaño que sea la mitad del original). Esto puede repetirse recursivamente hasta que se encuentre una ruta que atraviese únicamente celdas libres. (Por supuesto, el método sólo funciona si hay una forma de saber si una celda es una celda mixta, que es fácil sólo si los límites del espacio de configuración tienen descripciones matemáticas relativamente simples.) Este método es completo a condición de que haya un límite en el pasillo más pequeño a través del cual la solución deba pasar. A pesar de que se centra la mayoría del esfuerzo computacional en las áreas más difíciles dentro del espacio de configuración, sigue fallando el escalado en problemas de alta dimensionalidad porque cada división recursiva de una celda crea 2^d celdas más pequeñas. Una segunda forma de obtener un algoritmo completo es insistir en una **descomposición exacta en celdas** del espacio libre. Este método debe permitir que las celdas tengan formas irregulares de tal modo que estén en contacto con los límites del

DESCOMPOSICIÓN EXACTA EN CELDAS

espacio libre, pero las formas deben ser todavía «simples» en el sentido de que debería ser fácil calcular una travesía para cada celda libre. Esta técnica requiere algunas ideas geométricas algo avanzadas, así que no entraremos en más detalle.

Examinando la solución mostrada en la Figura 25.14(a), podemos ver dificultades adicionales que tienen que ser resueltas. Primero, se puede observar que la ruta contiene esquinas arbitrariamente agudas; un robot moviéndose a una velocidad finita no podría ejecutar una ruta de este tipo. Por otra parte, la ruta es muy cercana al obstáculo. Cualquiera que haya conducido un coche sabe que una plaza de aparcamiento con un milímetro de espacio en cada lado no es realmente una plaza de aparcamiento; por la misma razón, preferiríamos rutas solución que fueran robustas a pequeños errores de movimiento.

CAMPO DE POTENCIAL

Querríamos maximizar la distancia a los obstáculos mientras se minimiza la longitud de la ruta. Esto puede ser conseguido introduciendo un **campo de potencial**. Un campo de potencial es una función definida sobre el espacio de estados, cuyo valor crece con la distancia al obstáculo más cercano. La Figura 25.15(a) muestra un campo de potencial de este tipo (cuanto más oscuro sea el estado de configuración, más cercano estará a un obstáculo). Cuando se usa en la planificación de rutas, este campo de potencial se transforma en un término de coste adicional en la optimización. Esto introduce un cambio interesante. Por una parte, el robot busca minimizar la longitud del camino hasta el objetivo. Por otra parte, trata de mantenerse alejado de los obstáculos mediante la minimización de la función de potencial. Con el peso apropiado entre los dos objetivos, la ruta resultante puede parecerse a la mostrada en la Figura 25.15(b). Esta figura también muestra la función derivada de la función de coste combinada, calculada de nuevo mediante iteración de valores. Claramente, la ruta resultante es más larga, pero también es más segura.

Figura 25.15 (a) Un campo de potencial repulsivo empuja al robot lejos de los obstáculos. (b) La ruta encontrada mediante la minimización simultánea de la longitud de la ruta y el potencial.

Métodos de esqueletización

ESQUELETIZACIÓN

La segunda principal familia de algoritmos de planificación de rutas está basada en la idea de la **esqueletización**. Estos algoritmos reducen el espacio libre del robot a una representación unidimensional, para la cual el problema de planificación es más fácil. Esta representación de menor dimensionalidad es llamada **esqueleto** del espacio de configuración.

DIAGRAMA DE VORONOI

La Figura 25.16 muestra una esqueletización de ejemplo: es un **diagrama de Voronoi** del espacio libre (el conjunto de todos los puntos que son equidistantes de dos o más obstáculos). Para hacer planificación de rutas con un diagrama de Voronoi, el robot primero cambia su configuración presente a un punto en el grafo de Voronoi. Es fácil demostrar que esto siempre puede ser conseguido mediante un movimiento en línea recta en el espacio de configuración. Después, el robot sigue el diagrama de Voronoi hasta que alcanza el punto más cercano a la configuración objetivo. Finalmente, el robot deja el diagrama de Voronoi y se mueve al objetivo. Otra vez, el paso final implica movimiento en línea recta en el espacio de configuración.

De esta forma, el problema original de planificación de rutas se reduce a encontrar una ruta en el diagrama de Voronoi, que es generalmente unidimensional (excepto en casos no genéricos) y tiene varios puntos finitos donde tres o más curvas unidimensionales intersectan. De esta manera, encontrar el camino más corto en el diagrama de Voronoi es un problema de búsqueda en un grafo discreto del tipo de los mostrados en los Capítulos 3 y 4. Según el diagrama de Voronoi puede no darnos el camino más corto, pero las rutas resultantes tienden a maximizar el espacio libre. La desventaja principal de las técnicas basadas en diagramas de Voronoi es que son difíciles de aplicar a espacios de

(a) (b)

Figura 25.16 (a) El diagrama de Voronoi es un conjunto de puntos equidistantes a dos o más obstáculos en el espacio de configuración. (b) Un itinerario probabilístico, compuesto de 400 puntos elegidos al azar en espacio libre.

configuración de dimensiones más altas, y que tienden a inducir desvíos innecesariamente grandes cuando el espacio de configuración es muy abierto. Además, computar el diagrama de Voronoi puede ser difícil, específicamente en el espacio de configuración, donde la forma de los obstáculos puede ser compleja.

ITINERARIO PROBABILÍSTICO

Una alternativa a los diagramas de Voronoi es el **itinerario probabilístico**, una aproximación basada en esqueletización que ofrece más rutas posibles, y además funciona mejor con los espacios abiertos. La Figura 25.16(b) muestra un ejemplo de itinerario probabilístico. El grafo es creado generando aleatoriamente un gran número de configuraciones, y descartando aquellas que no caigan en espacio libre. Entonces, unimos un par de nodos cualquiera por un arco si es «fácil» alcanzar un nodo desde el otro, por ejemplo, por una línea recta en espacio libre. El resultado de todo esto es un grafo aleatorio en el espacio libre del robot. Si añadimos las configuraciones de inicio y objetivo del robot a este grafo, la planificación de ruta se traduce en la búsqueda en un grafo discreto. Teóricamente, esta aproximación es incompleta, ya que la mala elección de puntos aleatorios puede dejarnos sin ninguna ruta desde el inicio hasta el objetivo. Es posible limitar la probabilidad de fallo a partir del número de puntos generado y ciertas propiedades geométricas del espacio de configuración. También es posible dirigir la generación de puntos hacia áreas donde una búsqueda parcial sugiera que una buena ruta puede ser encontrada, trabajando bidireccionalmente desde las posiciones de inicio y final. Con estas mejoras, la planificación por itinerario probabilístico tiende a adaptarse mejor a espacios de configuración altamente dimensionados que muchas de las técnicas alternativas de planificación de rutas.

25.5 Planificar movimientos inciertos

Ninguno de los algoritmos de planificación de movimientos expuestos hasta ahora tiene en cuenta una de las características clave de los problemas en robótica: la *incertidumbre*. En robótica, la incertidumbre proviene de la observabilidad parcial del entorno y de los efectos estocásticos (o no modelados) de las acciones del robot. Los errores pueden también provenir del uso de algoritmos de aproximación como el filtrado de partículas, que no provee al robot con un estado de creencia exacto incluso si la naturaleza estocástica del entorno se modela perfectamente.

ESTADO MÁS PROBABLE

La mayoría de los robots de hoy en día usan algoritmos determinísticos para la toma de decisiones, como los algoritmos de planificación de rutas explicados hasta ahora. Para ello, es una práctica común extraer el **estado más probable** de la distribución de estados producida por el algoritmo de localización. La ventaja de esta aproximación es puramente computacional. Planificar rutas mediante espacios de configuración es todavía un problema desafiante; podría ser peor si tuviéramos que trabajar con una distribución de probabilidad plena sobre los estados. Ignorar la incertidumbre de esta manera funciona correctamente cuando la incertidumbre es pequeña.

Desafortunadamente, ignorar la incertidumbre no siempre funciona. En algunos problemas la incertidumbre del robot es simplemente demasiado grande. Por ejemplo, ¿cómo podríamos usar un planificador de rutas determinista para controlar un robot mó-

vil que no tiene idea de dónde está? En general, si el estado real del robot no es uno identificado por la regla de máxima probabilidad, el control resultante será subóptimo. Dependiendo de la magnitud del error esto puede llevar a toda una serie de efectos indeseables, como colisiones con obstáculos.

El campo de la robótica ha adoptado una gama de técnicas para acomodarse a la incertidumbre. Algunas se derivan de los algoritmos dados en el Capítulo 17 para la toma de decisiones bajo incertidumbre. Si el robot sólo se enfrenta a la incertidumbre en su estado de transición, pero este estado es plenamente observable, el problema es modelado mejor como un *proceso de Decisión de Markov*, o MDP. La solución para un MDP es una **política** óptima, que le dice al robot qué hacer en cada posible estado. De esta forma, puede manejar todos los tipos de errores de movimiento, mientras que una solución única obtenida mediante un planificador determinístico sería mucho menos robusta. En robótica, las políticas son normalmente llamadas **funciones de navegación**. La función mostrada en la Figura 25.14(a) puede ser convertida en una de estas funciones de navegación simplemente siguiendo el gradiente.

FUNCIONES DE NAVEGACIÓN

Como en el Capítulo 17, la observabilidad parcial hace el problema mucho más complicado. El problema de control del robot resultante es un **MDP observable parcialmente**, o POMDP. En estas situaciones, el robot normalmente mantiene un estado de creencia interno, como los discutidos en el Apartado 25.3. La solución a un POMDP es una política definida sobre el estado de creencia del robot. Dicho de otra forma, la entrada a la política es una distribución de probabilidad entera. Esto permite al robot basar sus decisiones no sólo en lo que conoce, sino también en lo que no. Por ejemplo, si no se tiene certeza sobre un estado crítico variable, se puede racionalmente invocar una **acción de recogida de información**. Esto es imposible en el marco del MDP, ya que los MDPs asumen observabilidad plena. Desafortunadamente, las técnicas que resuelven POMDPs de forma exacta son inaceptables para la robótica, no hay técnicas conocidas para espacios continuos. La discretización normalmente produce POMDPs que son demasiado grandes para ser manejados por técnicas conocidas. Todo lo que podemos hacer es intentar mantener la menor incertidumbre posible; por ejemplo, la heurística de **navegación costera** requiere que el robot permanezca cerca de marcadores conocidos para decrementar su incertidumbre de posicionamiento. Esto, además, decrementa gradualmente la incertidumbre de la obtención de nuevos marcadores que están cercanos, lo que permite al robot explorar más territorio.

ACCIÓN DE RECOGIDA DE INFORMACIÓN

NAVEGACIÓN COSTERA

Métodos robustos

ROBUSTOS

La incertidumbre puede ser también manejada utilizando los así llamados métodos **robustos** en lugar de los métodos probabilísticos. Un método robusto es aquel que asume una cantidad *limitada* de incertidumbre en cada aspecto del problema, pero que no asigna probabilidades a los valores en el interior del intervalo permitido. Una solución robusta es una que funciona sin importar cuáles son los valores actuales, siempre que se encuentren dentro del intervalo asumido. Una forma extrema de método robusto es la aproximación de **planificación conformista** dada en el Capítulo 12, produce planes que funcionan sin ningún tipo de información sobre el estado.

PLANIFICACIÓN DE MOVIMIENTO CERTERO

Aquí vemos un método robusto que es utilizado para la **planificación de movimiento certero** (*fine-motion planning* o FMP) en tareas de ensamblaje de robots. La planificación de movimiento certero implica mover un brazo robótico en una gran proximidad a un objeto estático del entorno. La principal dificultad con la planificación de movimiento certero es que los movimientos requeridos y las características relevantes del entorno son muy pequeños. A estas escalas tan pequeñas, el robot es incapaz de medir o controlar su posición de forma precisa y puede también no tener certeza sobre la forma del entorno; asumiremos que estas incertidumbres están todas limitadas. Las soluciones a los problemas de FMP serán típicamente planes condicionales o políticas que hagan uso de información de los sensores durante la ejecución y garantizan su buen funcionamiento en todas las situaciones consistentes con los límites asumidos de incertidumbre.

Un plan de movimiento certero consiste en una serie de **movimientos cautelosos**. Cada movimiento cauteloso consiste en (1) un comando de movimiento y (2) una condición de terminación, que es un predicado sobre el valor de los sensores del robot, y retorna verdadero para indicar el final del movimiento cauteloso. Los comandos de movimiento son típicamente **movimientos amoldables** que permiten al robot deslizarse si el comando de movimiento pudiera causar colisión con un obstáculo. Como un ejemplo, la Figura 25.17 muestra una configuración espacial en dos dimensiones con un estrecho agujero vertical. Podría ser el espacio de configuración para la inserción de una clavija rectangular en un agujero que es ligeramente más grande. Los comandos de movimiento son velocidades constantes. Las condiciones de terminación son el contacto con la superficie. Para modelar la incertidumbre en el control, asumimos que en lugar de mover en la dirección indicada, el movimiento real del robot permanece en el cono Cv. La figura muestra qué pasaría si ordenáramos una velocidad hacia abajo desde la región de inicio s. Debido a la incertidumbre en la velocidad, el robot podría moverse a cualquier lugar dentro de la región cónica, posiblemente dirigiéndose hacia el agujero, pero más probablemente situándose en algún lado del mismo. Debido a que el robot no sabría entonces en qué lado del agujero se encuentra, no sabría en qué dirección moverse.

Figura 25.17 Un entorno en dos dimensiones, cono de incertidumbre de velocidad, y región de posibles movimientos del robot. La velocidad buscada es v, pero con incertidumbre la velocidad real podría estar en cualquier lugar de Cv, resultando en una configuración final en cualquier lugar de la región de movimiento, lo que significa que no podríamos saber si hemos acertado en el agujero o no.

Una estrategia más sensata se muestra en las Figuras 25.18 y 25.19. En la Figura 25.18, el robot deliberadamente se mueve hacia uno de los lados del agujero. La orden de movimiento se muestra en la figura, y la condición de terminación es el contacto con alguna superficie. En la Figura 25.19, se da una orden de movimiento que causa que el robot se deslice a lo largo de la superficie y hacia el agujero. Esto asume que se utilice una orden de movimiento correcta. Debido a que todas las posibles velocidades en la región de movimiento son hacia la derecha, el robot se desplazará hacia la derecha siempre que se encuentre en contacto con una superficie horizontal. Se deslizará hacia el límite vertical inferior derecho del agujero cuando lo toque, porque todas las posibles velocidades relativas a una superficie vertical van hacia abajo. Se seguirá moviendo hasta que alcance la parte inferior del agujero, debido a que esa es su condición de terminación. A pesar de la incertidumbre en el control, todas las posibles trayectorias del robot terminan en contacto con la parte inferior del agujero, a menos que irregularidades en la superficie causen que el robot quede atascado en algún lugar.

Como podría imaginarse, el problema de *construir* planes de movimiento certero no es trivial; de hecho, es mucho más complicado que la planificación con movimientos exactos. Podemos o bien elegir un número fijo de valores discretos para cada movimiento o

Figura 25.18 La primera orden de movimiento y la región resultante de posibles movimientos del robot. No importa cuál sea el error, nosotros sabemos que la configuración final será a la izquierda del agujero.

Figura 25.19 La segunda orden de movimiento y la región de posibles movimientos. Incluso con error, a la larga llegaremos al agujero.

bien utilizar la geometría del entorno para elegir direcciones que den comportamientos cualitativamente diferentes. Un planificador de movimiento certero toma como entrada una descripción del espacio de configuración, el ángulo del cono de incertidumbre de la velocidad, y una especificación de qué estímulos son posibles para la terminación (contacto con la superficie en este caso). Produciría un plan condicional multipaso o una política de éxito garantizado, si dicho plan existiera.

Nuestro ejemplo asume que el planificador tiene un modelo exacto del entorno, pero es posible permitir un error acotado en este modelo como se indica a continuación. Si el error puede ser descrito en términos de parámetros, estos parámetros pueden ser añadidos como grados de libertad al espacio de configuración. En el último ejemplo, si la profundidad y la anchura del agujero fueran indeterminados, podríamos añadirlos como dos grados de libertad en el espacio de configuración. Es imposible mover al robot en estas direcciones en el espacio de configuración u obtener su posición directamente. Pero estas restricciones pueden ser incorporadas cuando se describa el problema como un problema FMP mediante la especificación de las incertidumbres de control y sensorización. Esto nos da un problema de planificación cuatridimensional y complejo, pero exactamente las mismas técnicas de planificación pueden ser aplicadas. Hay que notar que no muy diferente que los métodos de decisión teorética del Capítulo 17, este tipo de aproximación robusta resulta en planes diseñados para resolver el peor caso, más que para maximizar la calidad esperada de la planificación. Las planificaciones de peor caso son óptimas en el sentido de la decisión teorética si el fallo durante la ejecución es mucho peor que cualquier otro coste involucrado en la ejecución.

25.6 Movimiento

Hasta ahora, hemos hablado de *planificar* movimientos, pero no acerca de cómo *moverse*. Nuestros planes (particularmente aquellos producidos por planificadores de ruta determinísticos) asumen que el robot puede simplemente seguir cualquier ruta que el algoritmo produce. Esto no es cierto en el mundo real. Los robots tienen inercia y no pueden ejecutar rutas arbitrarias excepto a una baja velocidad. En la mayoría de los casos, más que especificar posiciones, el robot trata de aplicar fuerzas. Esta sección muestra métodos para calcular estas fuerzas.

Dinámica y control

El Apartado 25.2 introdujo la noción de **estado dinámico**, que extiende el estado cinemática de un robot mediante el modelado de sus velocidades. Por ejemplo, en adición al ángulo de la articulación de un robot, el estado dinámico además captura la velocidad de cambio del ángulo. El modelo de transición para la representación de un estado dinámico incluye el efecto de las fuerzas en esta velocidad de cambio. Estos modelos son típicamente expresados mediante **ecuaciones diferenciales**, que son ecuaciones que relacionan una cantidad (por ejemplo, un estado cinemático) con el cambio de la can-

ECUACIONES DIFERENCIALES

tidad con el tiempo (por ejemplo, velocidad). En principio podríamos haber elegido planificar el movimiento del robot usando modelos dinámicos, en lugar de nuestros modelos cinemáticos. Esta metodología nos llevaría a una actuación superior del robot, si pudiéramos generar los planes. Sin embargo, el estado dinámico es más complejo que el espacio cinemático, y la maldición de la dimensionalidad podría hacer los problemas de planificación de movimiento intratables para todos menos para los robots más simples. Por esta razón, los sistemas robóticos reales normalmente se basan en planificadores de rutas más simples.

CONTROLADOR

CONTROLADOR DE REFERENCIA

RUTA DE REFERENCIA

CONTROLADORES ÓPTIMOS

Una técnica común para compensar las limitaciones de las planificaciones cinemáticas es usar un mecanismo separado, un **controlador**, para mantener al robot en ruta. Los controladores son técnicas para generar controles del robot en tiempo real utilizando retroalimentación del entorno, así como para conseguir un objetivo de control. Si el objetivo es mantener al robot en una ruta preplaneada, entonces es común llamarlo **controlador de referencia** y a la ruta **ruta de referencia**. Los controladores que optimizan una función global de coste son conocidos como **controladores óptimos**. Las políticas óptimas para los MDPs son, en efecto, los controladores óptimos.

Superficialmente, el problema de mantener el robot en una ruta pre-especificada parece ser relativamente directo. En la práctica, sin embargo, incluso este problema aparentemente simple tiene sus trampas. La Figura 25.20(a) ilustra lo que puede ir mal. Lo que se muestra es una ruta para un robot que intenta seguir una ruta cinemática. Cada vez que se produce una desviación (ya sea por el ruido o por las restricciones en las fuerzas que el robot puede aplicar) el robot provee una fuerza en sentido contrario cuya magnitud es proporcional a su desviación. Intuitivamente, esto puede parecer factible, ya que las desviaciones podrían ser compensadas por una contra-fuerza para mantener al robot en el camino. Sin embargo, como muestra la Figura 25.20(a), nuestro controlador causa que el robot vibre más bien violentamente. La vibración es el resultado de la inercia natural del brazo robótico: una vez devuelto a su posición de referencia el robot puede llegar más allá, lo cual induce un error simétrico de signo opuesto. La Figura 25.20(a) muestra que esta desviación puede continuar a largo de la trayectoria completa, y que el movimiento resultante del robot estaría lejos de ser el deseable. Claramente hay una necesidad de un mejor control.

CONTROLADOR P

Para llegar a obtener un mejor controlador, describamos formalmente el tipo de controlador que produjo la desviación. Los controladores que proveen una fuerza en proporción negativa al error observado son conocidos como **controladores P**. La letra P viene de *proporcional*, indicando que el control actual es proporcional al error del manipulador del robot. Más formalmente, sea $y(t)$ la ruta de referencia, parametrizada por algún índice t. El control a_t generado por un controlador P tiene la siguiente forma:

$$a_t = K_P\,(y(t) - x_t)$$

PARÁMETRO DE GANANCIA

Aquí x_t es el estado del robot en el instante t. K_P es el llamado **parámetro de ganancia** del controlador que regula cuánto corrige la desviación el controlador entre el estado actual x_t y el deseado $y(t)$. En nuestro ejemplo, $K_p = 1$. En un primer momento, podemos pensar que elegir un valor menor de K_p remedia el problema. Desafortunadamente este no es el caso. La Figura 25.20(b) muestra una trayectoria para $K_p = ,1$, todavía describiendo un comportamiento oscilatorio. Valores menores del parámetro de ganancia pue-

Figura 25.20 Control de brazo robótica utilizando (a) control proporcional con factor de ganancia 1,0, (b) control proporcional con control de ganancia 0,1, y (c) control PD con factores de ganancia 0,3 para el componente proporcional y 0,8 para el diferencial. En todos los casos el robot trata de seguir la ruta mostrada en gris.

den simplificar la oscilación, pero no resolver el problema. De hecho, en ausencia de fricción, el controlador P es esencialmente una ley de resorte; por lo tanto oscilará indefinidamente alrededor de una localización objetivo fija.

Tradicionalmente, los problemas de este tipo se engloban dentro del ámbito de la **teoría de control**, un campo de importancia creciente para los investigadores en IA. Décadas de investigación en este campo han llevado a un mayor número de controladores que son superiores a la simple regla de control dada más arriba. En particular, un controlador de referencia se dice que es **estable** si pequeñas perturbaciones llevan a un error limitado entre el robot y la señal de referencia. Se dice que es **estrictamente estable** si es capaz de retornar a su ruta de referencia a pesar de tales perturbaciones. Claramente, nuestro controlador P parece ser estable pero no estrictamente estable, ya que no consigue retornar a su trayectoria de referencia.

ESTABLE

ESTRICTAMENTE ESTABLE

El controlador más simple que consigue estabilidad estricta en nuestro dominio se conoce como **controlador PD**. La letra 'P' sigue viniendo de *proporcional*, y 'D' viene de *derivativo*. Los controladores PD son descritos por la siguiente ecuación:

CONTROLADOR PD

$$a_t = K_P\,(y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t} \tag{25.3}$$

Como esta ecuación sugiere, los controladores PD extienden los controladores P con una componente diferencial, que añade al valor de a_t un término que es proporcional a la primera derivada del error $y(t) - x_t$ con respecto al tiempo. ¿Cuál es el efecto de este término? En general, un término de derivada amortigua el sistema que está siendo controlado. Para verlo, consideremos una situación donde el error $(y(t) - x_t)$ cambie rápidamente con el tiempo, como es el caso de nuestro controlador P indicado arriba. La derivada de este error contrarrestará entonces el término proporcional, lo que reducirá la respuesta global a la perturbación. Sin embargo, si el mismo error persiste y no cambia, la derivada se eliminará y entones el término proporcional dominará la elección del control.

La Figura 25.20(c) muestra el resultado de aplicar este controlador PD a nuestro brazo robótico, usando como parámetros de ganancia K_P = ,3 y K_D = ,8. Claramente la ruta resultante es mucho más suave, y no muestra ninguna oscilación obvia. Como este ejemplo sugiere, un término diferencial puede hacer estable un controlador que de otra forma no lo sería.

En la práctica, los controladores PD también pueden fallar en determinadas circunstancias. En particular, los controladores PD pueden fallar al regular un error próximo a cero, aun en ausencia de perturbaciones externas. Esto no es obvio en nuestro ejemplo, pero algunas veces se requiere una retroalimentación sobreproporcionada para disminuir un error hacia cero. La solución a este problema está en añadir un tercer término a la ley de control, basado en una integral del error a lo largo del tiempo:

$$a_t = K_P\,(y(t) - x_t) + K_I \int (y(t) - x_t)\,dt + K_D \frac{\partial(y(t) - x_t)}{\partial t} \tag{25.4}$$

Aquí K_I es otro parámetro de ganancia más. El término $\int (y(t) - x_t)\,dt$ calcula la integral del error con el tiempo. El efecto de este término es que las desviaciones duraderas entre la señal de referencia y el estado actual son corregidas. Si, por ejemplo, x_t es menor que $y(t)$ durante un largo período de tiempo, esta integral crecerá hasta que el control resultante a_t fuerza este error a reducirse. Los términos integrales, pues, aseguran que un controlador no muestra error sistemático, con la desventaja de un aumento en el riesgo del comportamiento oscilatorio. Un controlador con los tres términos es llamado **controlador PID**. Los controladores PID son muy usados en la industria, para una gran variedad de problemas de control.

CONTROLADOR PID

Control del campo de potencial

Introdujimos los campos de potencial como una función de coste adicional en la planificación de movimiento del robot, pero también pueden ser usados para generar movimientos del robot directamente, en conjunto con la fase de planificación de movimiento. Para conseguirlo, debemos definir una fuerza atractiva que empuje al robot hacia su configuración objetivo y un campo potencial repulsivo que empuje al robot lejos de los obstáculos. Este tipo de campo de potencial se muestra en la Figura 25.21. Su único mínimo global es la configuración objetivo, y el valor es la suma de la distancia a su configuración objetivo y la proximidad a los obstáculos. Ninguna planificación está involucrada en la generación del campo de potencial mostrado en la figura. Debido a esto, los campos de potencial son muy adecuados para control en tiempo real. La Figura 25.21 muestra dos trayectorias de un robot que realiza el ascenso de una elevación en el campo de potencial, a partir de dos configuraciones iniciales diferentes. Además, optimizando las cantidades de potencial mediante el cálculo del gradiente de potencial para la configuración actual del robot. Estos cálculos normalmente son extremadamente eficientes, especialmente cuando se comparan con los algoritmos de planificación de trayectorias, los cuales son exponenciales en la dimensionalidad del espacio de configuración (los grados de libertad).

Figura 25.21 Control por campo de potencial. El robot asciende por un campo de potencial compuesto de fuerzas de repulsión obtenidas a partir de los obstáculos, y fuerzas atrayentes que corresponden a la configuración objetivo. (a) Ruta con éxito. (b) Óptimo local.

El hecho de que la aproximación del campo de potencial trate de encontrar una ruta al objetivo de esta forma tan eficiente, a pesar de largas distancias en el espacio de configuración, hace surgir la pregunta de por qué es necesaria la planificación en robótica. ¿Son los campos de potencial suficientes o simplemente hemos sido afortunados en nuestro ejemplo? La respuesta es que ciertamente hemos sido afortunados. Los campos de potencial tienen varios mínimos locales que pueden atrapar al robot. En este ejemplo, el robot se aproxima al obstáculo simplemente rotando su articulación de hombro, hasta que queda atrapado en el lado equivocado del obstáculo. El campo de potencial no es lo suficientemente rico para hacer al robot flexionar su codo de tal forma que el brazo quepa bajo el obstáculo. En otras palabras, las técnicas de campo de potencial son muy buenas para control local del robot pero todavía necesitan planificación global. Otra desventaja importante de los campos de potencial es que las fuerzas que generan dependen sólo de la posición del robot y de los obstáculos, no de la velocidad del robot. Por lo tanto, el control por campo de potencial realmente es un método cinemático y puede fallar si el robot se mueve rápidamente.

Control reactivo

Hasta ahora hemos considerado decisiones de control que requieren algún modelo del entorno para construir o bien una ruta de referencia o bien un campo de potencial. Hay algunas dificultades con esta aproximación. Primero, los modelos que son lo suficientemente precisos son a menudo difíciles de obtener, especialmente en entornos complejos o remotos, como la superficie de Marte. Segundo, incluso en los casos en los que poda-

mos idear un modelo con la suficiente precisión, las dificultades computacionales y el error de localización pueden hacer estas técnicas nada prácticas. En algunos casos, un diseño de agentes guiados por reflejos (el llamado **control reactivo**) es más apropiado.

CONTROL REACTIVO

HEXÁPODO

Uno de estos ejemplos es el robot de seis piernas, o **hexápodo** mostrado en la Figura 25.22(a), que tiene la tarea de andar a través de terreno abrupto. Los sensores del robot son enormemente inadecuados para obtener modelos del terreno con la suficiente precisión para que cualquiera de las técnicas de planificación descritas en las secciones previas funcione. Además, incluso si añadiésemos los suficientes sensores precisos, los doce grados de libertad (dos por cada pierna) harían del problema de planificación de ruta resultante algo computacionalmente intratable.

Es posible, aun así, especificar un controlador directamente sin un modelo explícito del entorno. (Hemos visto ya esto con el controlador PD, que era capaz de dirigir un brazo robótico al objetivo *sin* un modelo explícito de la dinámica del robot; ello requería, sin embargo, una ruta de referencia generada a partir de un modelo cinemático.) Para el ejemplo de nuestro robot con piernas, especificar una ley de control se convierte en algo sorprendentemente simple en el nivel de abstracción correcto. Una ley de control viable podría hacer moverse cada pierna en ciclos, de tal forma que parte del tiempo se encuentre tocando el suelo y otra parte del tiempo permanezca en el aire. Las seis piernas deberían estar coordinadas de tal forma que tres de ellas (en extremos opuestos) estuvieran siempre en tierra para proveer de soporte físico. Este patrón de control es fácilmente programable y funciona muy bien en terreno plano. En terreno abrupto, los obstáculos pueden evitar a las piernas oscilar hacia delante. Este problema puede ser solucionado con una regla de control extremadamente simple: *cuando el movimiento hacia delante de una pierna esté bloqueado, simplemente retrásala, elévala más alto, y prueba de nuevo*. El controlador resultante se muestra en la Figura 25.22(b) como un autómata finito; constituye un agente guiado por reflejos con estado, donde el estado interno está representado por el índice del estado actual de la máquina (de s_1 a s_4).

(a) (b)

Figura 25.22 (a) un robot hexápodo (b) Un autómata finito aumentado (AFSM o *augmented finite state machine*) para el control de una única pierna. Este AFSM reacciona a la retroalimentación de los sensores: si una pierna se atasca durante la fase de movimiento hacia delante, se elevará incrementalmente.

Variantes de este simple controlador guiado por retroalimentación pueden ser encontradas para generar un patrón robusto para caminar, capaz de hacer maniobrar al robot a través de terreno abrupto. Claramente, dicho controlador no se ajusta a ningún modelo, y no hace uso de deliberación o de búsqueda para generar controles. Cuando se ejecuta un controlador de este tipo, la retroalimentación del entorno representa un papel crucial en el comportamiento generado por el robot. El *software* por sí mismo no especifica qué pasará cuando el robot se coloque en el entorno. El comportamiento que surge por la interactuación entre un controlador (simple) y un entorno (complejo) es normalmente llamado **comportamiento emergente**. Estrictamente hablando, todos los robots mencionados en este capítulo muestran un comportamiento emergente, debido a que no existe modelo perfecto. Históricamente, sin embargo, el término se ha reservado para técnicas de control que no utilizan modelos explícitos del entorno. El comportamiento emergente es también característico de un gran número de organismos biológicos.

COMPORTAMIENTO EMERGENTE

Técnicamente, los controladores reactivos son sólo una implementación de una política de un MDP (o, si tienen estado interno, de un POMDP). En el Capítulo 17, encontrábamos diversas técnicas para generar políticas a partir de modelos del robot y su entorno. En robótica, crear este tipo de políticas a mano es de gran importancia práctica, debido a nuestra incapacidad para formular modelos precisos. El Capítulo 21 describía métodos de refuerzo de aprendizaje para construir políticas a partir de la experiencia. Algunos de estos métodos (como el Q-*learning* y métodos de búsqueda de políticas) no requieren modelo del entorno y son capaces de generar controladores de alta calidad para los robots, pero a costa de depender de una gran cantidad de datos de entrenamiento.

25.7 Arquitecturas *software* robóticas

ARQUITECTURA *SOFTWARE*

Una metodología para estructurar algoritmos es la llamada **arquitectura *software***. Una arquitectura normalmente incluye lenguajes y herramientas para escribir programas, así como una filosofía global de cómo los programas deben unirse.

Las arquitecturas *software* actuales para la robótica deben decidir cómo combinar control reactivo y control deliberativo basado en modelos. De muchas maneras, el control reactivo y el deliberativo tienen fuerzas y debilidades ortogonales. El control reactivo es guiado por sensores y apropiado para hacer decisiones de bajo nivel en tiempo real. Sin embargo, el control reactivo raramente nos da una solución plausible a un nivel global, debido a que las decisiones de control global dependen de información que no puede ser obtenida durante el período de toma de decisiones. Para este tipo de problemas, el control deliberativo es más apropiado.

Consecuentemente, la mayoría de las arquitecturas robóticas usan técnicas reactivas en los niveles más bajos de control con técnicas deliberativas en los niveles más altos. Encontramos una combinación de este tipo en nuestra explicación de los controladores PD, donde combinamos un controlador PD (reactivo) con un planificador de rutas (deliberativo). Las arquitecturas que combinan técnicas reactivas y deliberativas son normalmente llamadas **arquitecturas híbridas**.

ARQUITECTURAS HÍBRIDAS

Arquitectura de subsumpción

ARQUITECTURA DE SUBSUMPCIÓN

La **arquitectura de subsumpción** (Brooks, 1986) es una plataforma para ensamblar controladores reactivos en autómatas finitos. Los nodos de estos autómatas pueden contener chequeos para ciertas variables de sensores, en cuyo caso la ejecución del autómata finito está condicionada por el resultado de dicho chequeo. Los arcos pueden ser etiquetados con mensajes que se generarán cuando se atraviesen, y que son enviados a los motores del robot o a otros autómatas finitos. Adicionalmente, los autómatas finitos poseen temporizadores internos (relojes) que controlan el tiempo que se necesita para atravesar un arco. Los autómatas resultantes son normalmente conocidos como **autómatas finitos aumentados**, o AFSMs (*augmented finite state machines*), donde el aumento se refiere al uso de relojes.

AUTÓMATAS FINITOS AUMENTADOS

Un ejemplo de un AFSM simple es el autómata de cuatro estados mostrado en la Figura 25.22(b), que genera un movimiento cíclico de las piernas de un robot hexápodo. Este AFSM implementa un controlador cíclico, cuya ejecución mayormente no se basa en la retroalimentación del entorno. Si la pierna queda atascada, lo cual significa que ha fallado en ejecutar el movimiento hacia delante, el robot retrasa la pierna, la eleva un poco más alto, y trata de realizar el movimiento hacia delante otra vez. De esta manera, el controlador es capaz de *reaccionar* a contingencias surgidas de la interacción entre el robot y su entorno.

La arquitectura de subsumpción ofrece primitivas adicionales para sincronizar AFSMs, y para combinar valores de salida de múltiples y posiblemente conflictivos AFSMs. De esta forma, permite al programador componer controladores más complejos de forma incremental de una forma ascendente o de abajo arriba. En nuestro ejemplo, podríamos empezar con AFSMs para piernas individuales, seguido por un AFSMs para coordinar las múltiples piernas. Sobre esto podríamos implementar comportamientos de más alto nivel como un evitador de obstáculos, que podría implicar retroceder y girar.

La idea de componer controladores robóticos a partir de AFSMs es bastante desconcertante. Imaginemos por un momento qué difícil sería generar el mismo comportamiento con alguno de los algoritmos de planificación de rutas descritos en la sección anterior. Primero, necesitaríamos un modelo preciso del terreno. El espacio de configuración del robot con seis piernas, cada una de las cuales es conducida por dos motores individuales, tiene en total 18 dimensiones (12 dimensiones para las configuraciones de las piernas, y seis para la localización y orientación del robot relativa a su entorno). Incluso si nuestros computadores fueran lo suficientemente rápidos para encontrar rutas en espacios tan sobredimensionados, deberíamos preocuparnos de efectos desagradables como el robot resbalando por una cuesta. Debido a tales efectos estocásticos, una ruta a través del espacio de configuración sería demasiado frágil, e incluso un controlador PID podría no ser capaz de adaptarse a estas contingencias. En otras palabras, generar comportamientos de movimiento deliberativamente es simplemente un problema demasiado complejo para los algoritmos de planificación de movimiento de robots de hoy en día.

Desafortunadamente, la arquitectura de subsumpción también tiene problemas. Primero, los AFSMs son normalmente conducidos por la entrada de los sensores en bruto,

una aproximación que funciona si los datos de los sensores son fidedignos y contienen toda la información necesaria para la toma de decisiones, pero falla si los datos de los sensores deben ser integrados de forma no trivial con el tiempo. Los controladores de estilo de subsumpción han sido hasta aplicados sobre todo en tareas locales, como el seguimiento de una pared o moverse hacia fuentes de luz visibles. Segundo, la falta de deliberación hace difícil cambiar la tarea del robot. Un robot guiado por subsumpción normalmente hace una tarea, y no tiene noción de cómo modificar sus controles para acomodarse a los diferentes objetivos de control (como el escarabajo del Apartado 2.2). Finalmente, los controladores de subsumpción suelen ser difíciles de entender. En la práctica, la intrincada interacción entre docenas de AFSMs (y el entorno) está más allá de lo que la mayoría de los programadores humanos pueden entender. Por todas estas razones, la arquitectura de subsumpción es raramente usada en robótica comercial, a pesar de su gran importancia histórica. Sin embargo, algunos de sus descendientes sí son utilizados.

Arquitectura de tres capas

ARQUITECTURA DE TRES CAPAS

Las arquitecturas híbridas combinan reacción con deliberación. De lejos, la arquitectura híbrida más popular es la **arquitectura de tres capas**, que consiste en una capa reactiva, una capa ejecutiva y una capa deliberativa.

CAPA REACTIVA

La **capa reactiva** provee de control de bajo nivel al robot. Se caracteriza por fuerte bucle sensor-acción. Su ciclo de decisión se encuentra normalmente en el orden de los milisegundos.

CAPA EJECUTIVA

La **capa ejecutiva** (o capa encadenadora) sirve como unión entre las capas reactiva y deliberativa. Acepta directivas de la capa deliberativa, y las encadena para la capa reactiva. Por ejemplo, la capa ejecutiva podría manejar un conjunto de puntos de ruta generados por un planificador de rutas deliberativo, y tomar decisiones como por ejemplo qué comportamiento reactivo invocar. Los ciclos de decisión en la capa ejecutiva se encuentran normalmente en el orden del segundo de tiempo. La capa ejecutiva es también responsable de integrar la información de los sensores en una representación interna del estado. Por ejemplo, puede almacenar la localización del robot y rutinas de mapeado.

CAPA DELIBERATIVA

La **capa deliberativa** genera soluciones globales para tareas complejas utilizando planificación. Debido a la complejidad computacional necesaria para la generación de tales soluciones, su ciclo de decisión suele encontrarse en el rango de varios minutos. La capa deliberativa (o capa planificadora) usa modelos para la toma de decisiones. Estos modelos pueden ser presuministrados o aprendidos de los datos, y normalmente utilizan información de estado recolectada en la capa ejecutiva.

Variantes de la arquitectura de tres capas pueden ser encontradas en la mayoría de los sistemas *software* robóticos de hoy en día. La descomposición en tres capas no es muy estricta. Algunos sistemas *software* robóticos poseen capas adicionales, como capas de interfaz de usuario que controlan la interacción con la gente, o capas responsables de coordinar las acciones del robot con aquellas de otros robots operando en el mismo entorno.

Lenguajes de programación robóticos

LENGUAJE DE COMPORTAMIENTOS

Algunos controladores robóticos han sido implementados con lenguajes de programación de propósito especial. Por ejemplo, varios programas para la arquitectura de subsumpción han sido implementados en el **lenguaje de comportamientos** (*behavior language*) definido por Brooks (1990). Este lenguaje es un lenguaje de control basado en reglas y en tiempo real que compila controladores AFSM. Las reglas individuales en una sintaxis parecida a Lisp son compiladas en AFSMs, y múltiples AFSMs son integrados mediante una colección de mecanismos de paso de mensajes locales y globales.

Como ocurre con la arquitectura de subsumpción, el lenguaje de comportamientos está limitado a AFSMs simples con una definición relativamente estrecha del flujo de comunicación entre módulos. La investigación reciente se ha construido sobre esta idea, llevando a un conjunto de lenguajes de programación similares en esencia al lenguaje de comportamientos, pero más poderosos y rápidos en ejecución. Uno de estos lenguajes es el **lenguaje robótico genérico** (*generic robot language* o GRL, Horswill, 2000). GRL es un lenguaje de programación funcional para programar grandes sistemas de control modulares. Como en el lenguaje de comportamientos, GRL usa autómatas finitos como bloques básicos de construcción. Además de esto, provee un rango de constructores más extenso para definir flujos de comunicación y restricciones de sincronización entre diferentes módulos que el lenguaje de comportamientos. Los programas en GRL son compilados en lenguajes imperativos más eficientes, como C.

LENGUAJE ROBÓTICO GENÉRICO

Otro lenguaje de programación importante (y arquitectura asociada) para *software* robótica concurrente es el sistema de planificación de acción reactiva (*reactive action plan system*, o RAPS, Firby, 1994). RAPS permite a los programadores especificar objetivos, planes (o políticas parciales) asociados con estos objetivos, y condiciones bajo las que estos planes podrían tener éxito. Fundamentalmente, RAPS también provee facilidades para manejar los *fallos* inevitables que se producen con los sistemas robóticos reales. El programador puede especificar rutinas de decisión para varios tipos de fallo y proporcionar una rutina de manejo de excepciones para cada tipo. En arquitecturas de tres capas, RAPS es normalmente usado en la capa ejecutiva, para manejar contingencias que no requieren replanificación.

Hay otros lenguajes que permiten que el razonamiento y el aprendizaje tenga lugar en el robot. Por ejemplo, GOLOG (Levesque *et al.*, 1997b) es un lenguaje de programación que de un modo transparente mezcla resolución deliberativa de problemas (planificación) y especificación directa de control reactivo. Los programas en GOLOG son formulados en cálculo de situación (Apartado 10.3), con la opción adicional de operadores de acción no deterministas. En adición a la especificación de un programa de control con posibles acciones no deterministas, el programador puede también tener que proporcionar un modelo completo del robot y del entorno. Cada vez que el programa de control alcanza un punto de elección no determinista, un planificador (en la forma de demostrador de teoremas) es invocado para determinar qué hacer a continuación. De esta forma, el programador puede especificar controladores parciales y confiar en planificadores incorporados para formar la decisión final de control. La belleza de GOLOG está en su integración transparente de reactividad y deliberación. A pesar de los enormes re-

querimientos de GOLOG (observabilidad plena, estados discretos, modelo completo), GOLOG ha proporcionado control de alto nivel para una serie de robots móviles de interior.

CES

CES, siglas de C++ para sistemas empotrados (C++ *for embedded systems*), es un lenguaje que extiende C++ e integra probabilidades y aprendizaje (Thrun, 2000). Los tipos de datos en CES son distribuciones de probabilidad, permitiendo al programador calcular con información imprecisa sin el esfuerzo normalmente requerido para implementar técnicas probabilísticas. Y lo que es más importante, CES hace posible entrenar el *software* robótico con ejemplos, muy parecido a los algoritmos de aprendizaje descritos en el Capítulo 20. CES permite a los programadores dejar «huecos» en el código que son rellenados por funciones aprendibles, típicamente representaciones paramétricas diferenciales como redes neuronales. Estas funciones son entonces aprendidas inductivamente en fases de entrenamiento explícitas, donde el entrador debe especificar el comportamiento de salida deseado. CES ha demostrado trabajar bien en dominios continuos y parcialmente observables.

ALISP

Alisp (Andre y Russell, 2002) es una extensión de Lisp. ALisp permite a los programadores especificar puntos de elección no deterministas, similarmente a los puntos de decisión de GOLOG. Sin embargo, en lugar de confiar en un comprobador de teoremas para tomar las decisiones, ALisp aprende inductivamente la acción correcta mediante aprendizaje de refuerzo. De esta manera, ALisp puede ser visto como un medio flexible para incorporar conocimiento del dominio (especialmente conocimiento acerca de la estructura jerárquica de los comportamientos deseados) en un sistema de aprendizaje por refuerzo. Hasta ahora ALisp sólo ha sido aplicado a problemas robóticos en simulación, pero proporciona una metodología prometedora para construir robots que aprenden a través de la interacción con el entorno.

25.8 Dominios de aplicación

Ahora enumeraremos algunos de los dominios de aplicación principales para la tecnología robótica.

Industria y agricultura. Tradicionalmente, los robots han sido utilizados en áreas que requieren un trabajo difícil por parte de los humanos, aunque están lo suficientemente estructuradas para ser manejadas por automatización robótica. El mejor ejemplo es la línea de ensamblaje, donde los manipuladores de forma rutinaria realizan tareas como ensamblar, colocar piezas, manejar materiales, soldar y pintar. En algunas de estas tareas, los robots se han hecho más rentables que los trabajadores humanos.

En el exterior, algunas de las máquinas pesadas que se utilizan para cosechar, minar, o excavar la tierra han sido transformadas en robots. Por ejemplo, un proyecto reciente de Carnegie Mellon ha demostrado que los robots pueden eliminar la pintura de grandes barcos alrededor de 50 veces más rápido que las personas, y con un impacto ambiental mucho más reducido. Prototipos de robots mineros autónomos son más rápidos y precisos que los humanos para el transporte de mineral en minas subterráneas. Los robots han sido utilizados para generar mapas de alta precisión de minas abandonadas y

sistemas de alcantarillado. Mientras que muchos de estos sistemas son todavía prototipos, es sólo cuestión de tiempo que los robots asuman la mayoría del trabajo semi-mecánico llevado a cabo actualmente por los humanos.

Transporte. El transporte robótico tiene varias facetas: desde helicópteros autónomos que entregan objetos a localizaciones que podrían ser difíciles de acceder por otros medios, a sillas de ruedas automáticas que transportan gente que es incapaz de controlarlas por sí misma, o portadores de carga autónomos que superan a los humanos experimentados al transportar contenedores desde barcos a camiones en muelles de carga. Un primer ejemplo de robot para transporte en el interior, o recadero, es el robot Helpmate mostrado en la Figura 25.23(a). Este robot ha sido desplegado en docenas de hospitales para transportar comida y otros objetos. Los investigadores han desarrollado sistemas robóticos con forma de coche que pueden navegar autónomamente en autopistas o a través de terreno externo a la carretera. En entornos industriales, los vehículos autónomos son desplegados rutinariamente para transportar bienes en almacenes y entre líneas de producción.

Algunos de estos robots requieren modificaciones en su entorno para poder realizar sus funciones. Las modificaciones más comunes son ayudas de localización como bucles inductivos en el suelo, faros activos, etiquetas de códigos de barras y satélites GPS. Un reto abierto en robótica es el diseño de robots que puedan usar pistas naturales, en lugar de dispositivos artificiales, para navegar, sobre todo en entornos como el océano profundo donde no puede ser utilizado GPS.

Entornos peligrosos. Los robots han ayudado a la gente a limpiar residuos nucleares, muy notablemente en Chernobyl y Three Mile Island. Los robots estuvieron presentes tras el colapso del World Trade Center, donde entraron en estructuras aparentemente demasiado peligrosas para la búsqueda humana y grupos de rescate.

(a)

(b)

Figura 25.23 (a) El robot Helpmate transporta comida y otros elementos médicos en docenas de hospitales alrededor del mundo. (b) Robots quirúrgicos en la sala de operaciones (por da Vinci Surgical Systems).

Algunos países han utilizado robots para transportar munición y desactivar bombas (una tarea notablemente peligrosa). Un número de proyectos de investigación están actualmente desarrollando prototipos robóticos para limpiar campos de minas, en tierra o por mar. La mayoría de los robots existentes para estas tareas son teleoperador, un humano los opera mediante control remoto. Proveer a este tipo de robots de autonomía es el siguiente paso importante.

Exploración. Los robots han ido a donde nadie ha ido antes, incluyendo la superficie de Marte. (*Véase* Figura 25.1(a).) Los brazos robóticos asisten a los astronautas en la colocación y recogida de satélites y la construcción de la Estación Espacial Internacional. Los robots también ayudan a explorar por debajo del mar. Son rutinariamente utilizados para obtener mapas de barcos hundidos. La Figura 25.24 muestra un robot realizando un mapa de una mina de carbón abandonada, mediante un modelo 3-D de la mina obtenido mediante sensores de rango. En 1996, un equipo de investigadores soltaron un robot con piernas en el cráter de un volcán activo para obtener datos importantes para la investigación climática. Vehículos aéreos no tripulados conocidos como **drones** son usados para operaciones militares. Los robots están pasando a ser herramientas muy efectivas para recolectar información en dominios que son difíciles (y peligrosos) de acceder para la gente.

DRONES

Salud. Los robots están siendo usados cada vez más para ayudar a los cirujanos en la colocación de instrumentos cuando operan con órganos tan interrelacionados como el cerebro, ojos y corazón. La Figura 25.23(b) muestra un sistema de este tipo. Los robots han pasado a ser herramientas indispensables en ciertos tipos de reemplazos de caderas, gracias a su gran precisión. En estudios piloto, se ha comprobado que los dispositivos robóticos reducen el riesgo de lesión cuando se realizan colonoscopias. Fuera de la sala de operaciones, los investigadores han empezado a desarrollar ayudas robóticas para la gente anciana o discapacitada, como andadores robóticos inteligentes y juguetes inteligentes que recuerdan cuándo se debe tomar la medicación.

Servicios personales. Los servicios son una aplicación reciente en el dominio de la robótica. Los robots de servicio ayudan a los individuos a realizar tareas diarias. Los robots de servicio domésticos comercialmente disponibles incluyen aspiradoras autóno-

(a) (b)

Figura 25.24 (a) Un robot creando un mapa de una mina abandonada de carbón. (b) Un mapa 3-D de la mina obtenido por el robot.

mas, cortacéspedes, y caddies de golf. Todos estos robots pueden navegar autónomamente y realizar sus tareas sin ayuda humana. Algunos robots de este tipo operan en lugares públicos, como los quioscos de información robóticos que han sido emplazados en centros comerciales y ferias de comercio, o en museos como guías turísticos. Las tareas de servicio requieren interacción humana, y la habilidad de enfrentarse robustamente con entornos impredecibles y dinámicos.

Entretenimiento. Los robots han comenzado a conquistar la industria de los juguetes y del entretenimiento. Vimos el Sony AIBO en la Figura 25.4(b); este juguete robótico en forma de perro está empezando a ser utilizado como plataforma de investigación en los laboratorios de IA alrededor del mundo. Una tarea de IA desafiante que se estudia con esta plataforma es el **fútbol robótico**, una competición muy parecida al fútbol humano, pero jugado por robots móviles autónomos. El fútbol robótico provee de grandes oportunidades para la investigación en la IA, ya que plantea toda una gama de problemas prototipo para muchas otras aplicaciones robóticas más serias. Las competiciones de fútbol robótico anuales han atraído a un gran número de investigadores de IA y añadido mucha animación al campo de la robótica.

FÚTBOL ROBÓTICO

Aumento humano. Un dominio de aplicación final para la tecnología robótica es el aumento humano. Los investigadores han desarrollado máquinas con piernas andantes que pueden transportar gente, como si se tratara de sillas de ruedas. Varios esfuerzos de investigación actualmente se centran en el desarrollo de dispositivos que hagan a la gente andar o mover sus brazos, y proveen de fuerzas adicionales mediante añadidos extraesqueléticos. Si dichos dispositivos son fijados permanentemente, se podría pensar en ellos como miembros robóticos artificiales. La teleoperación robótica implica llevar a cabo tareas a través de largas distancias, con la ayuda de dispositivos robóticos. Una configuración popular para la teleoperación robótica es la configuración maestro-esclavo, en la cual un manipulador robótico emula el movimiento de un operador humano robótico, medido a través de una interfaz táctil. Todos estos sistemas aumentan la habilidad de la gente para interactuar con sus entornos. Algunos proyectos intentan incluso replicar humanos, al menos a un nivel muy superficial. Los robots humanoides están disponibles comercialmente en estos momentos gracias a varias compañías en Japón.

25.9 Resumen

El interés de la robótica se centra en los agentes inteligentes que manipulan el mundo físico. En este capítulo, hemos aprendido los siguientes conceptos básicos sobre el *hardware* y el *software* robóticos.

- Los robots están equipados con sensores para percibir su entorno y efectores con los que pueden aplicar fuerzas físicas en su entorno. La mayoría de los robots son o bien manipuladores anclados en posiciones fijas o robots móviles que se pueden mover.
- La percepción robótica trata de determinar cantidades relevantes para la decisión a partir de los datos de los sensores. Para ello, necesitamos una representación in-

terna y un método para actualizar su representación interna a lo largo del tiempo. Ejemplos comunes de problemas perceptuales incluyen localización y elaboración de mapas.

- Los algoritmos de filtrado probabilístico como los filtros de Kalman y filtros de partículas son útiles para la percepción de los robots. Estas técnicas mantienen el estado de creencia, por ejemplo, una distribución posterior sobre las variables de estado.
- La planificación del movimiento de un robot es normalmente hecha en el espacio de configuración, donde cada punto especifica la localización y la orientación del robot y los ángulos de sus articulaciones.
- Los algoritmos de búsqueda en los espacios de configuración incluyen técnicas de descomposición en celdas, que descomponen el espacio de todas las configuraciones en varias celdas finitas, y técnicas de esqueletización, que proyectan espacios de configuración en estructuras de menos dimensiones. El problema de planificación de movimiento se resuelve entonces utilizando búsqueda en estas estructuras más simples.
- Una ruta encontrada por un algoritmo de búsqueda puede ser ejecutada utilizando dicha ruta como trayectoria de referencia para controladores PID.
- Las técnicas de campo de potencial hacen navegar a los robots mediante funciones de potencial, definidas según la distancia a los obstáculos y a la localización objetivo. Las técnicas de campo de potencial pueden quedar estancadas en un mínimo local, pero pueden generar movimiento directamente sin necesidad de planificación de rutas.
- Algunas veces, es más sencillo especificar un controlador robótico directamente, más que derivar una ruta desde un modelo explícito del entorno. Dichos controladores pueden ser a menudo escritos como simples autómatas finitos.
- La arquitectura de subsumpción permite a los programadores componer los controladores robóticos a partir de autómatas finitos interconectados, aumentados mediante temporizadores incorporados.
- Las arquitecturas de tres capas son marcos comunes para el desarrollo de *software* robótico que integre deliberación, secuenciación de subobjetivos y control.
- Existen lenguajes de programación de propósito general que facilitan el desarrollo de *software* robótico. Estos lenguajes proporcionan construcciones para desarrollar *software* multi-hilo, para integrar directivas de control en la planificación, y para aprender de la experiencia.

Notas bibliográficas e históricas

La palabra **robot** fue popularizada por el dramaturgo checo Karen Capek en su obra de 1921 *R.U.R.* (Rossum's Universal Robots). Los robots, que crecían químicamente en lugar de ser construidos mecánicamente, terminaron sintiendo resentimiento por sus maestros y decidiendo asumir el control. Parece ser (Glanc, 1978), que de hecho fue el hermano de Capek, Josef, quien primero combinó las palabras «robota» (trabajo obligatorio) y «robotnik» (siervo) para producir «robot» en su historia corta de 1917 *Opilec*.

El término *robótica* fue usado por primera vez por (Asimov, 1950). La robótica (bajo otros nombres), tiene una historia mucho más larga, sin embargo. En la mitología griega, un hombre mecánico llamado Talos fue supuestamente diseñado y construido por Hephaistos, el dios griego de la metalurgia. Se construyeron autómatas maravillosos durante el siglo XVIII (El pato mecánico de Jacques Vaucanson de 1738 es un ejemplo temprano) pero los comportamientos complejos que exhibían se mantenían solamente en el avance. Posiblemente, el ejemplo más antiguo de dispositivo programable parecido a un robot fue el telar de Jacquard (1805), descrito en el Capítulo 1.

UNIMATE

El primer robot comercial fue un brazo robótico llamado **Unimate**, abreviatura de *universal automation*. Unimate fue desarrollado por Joseph Engelberger y George Devol. En 1961, el primer robot Unimate se vendió a General Motors, donde fue utilizado para fabricar tubos de televisión. 1961 fue también el año en que Devol obtuvo la primera patente en Estados Unidos relacionada con los robots. 11 años más tarde, en 1972, Nissan Corp. fue la primera compañía que automatizó una cadena entera de ensamblaje con robots, desarrollados por Kawasaki con robots suministrados por la compañía Unimation de Engelberger y Devol. Este desarrollo inició una revolución mayor que tuvo lugar principalmente en Japón y Estados Unidos, y que todavía continúa. Unimation repitió en 1978 con el desarrollo del robot **PUMA**, abreviatura de Programable Universal Machine for Assembly. El robot PUMA, inicialmente desarrollado por General Motors, fue el estándar *de facto* para la manipulación robótica en las dos décadas siguientes. En el presente, el número de robots operando se estima de un millón alrededor de todo el mundo, más de la mitad de los cuales están instalados en Japón.

PUMA

La literatura sobre la investigación en robótica puede ser dividida a grandes rasgos en dos partes: robots móviles y manipuladores estáticos. El «tortuga»de Grey Walter, construido en 1948, puede ser considerado el primer robot móvil autónomo, a pesar de que su sistema de control no era programable. El «Hopkins Beast», construido en los primeros 60 en la Universidad Johns Hopkins, era mucho más sofisticado; tenía un *hardware* de reconocimiento de patrones y podía reconocer la cubierta protectora de un enchufe estándar AC. ¡Era capaz de buscar enchufes, enchufarse en ellos, y entonces recargar sus baterías! Sin embargo, el Beast tenía un repertorio limitado de habilidades. El primer robot de propósito general fue «Shakey», desarrollado en lo que era entonces el Stanford Research Institute (ahora SRI) al final de los 60 (Fikes y Nilsson, 1971; Nilsson, 1984). Shakey fue el primer robot que integraba percepción, planificación y ejecución, y mucha de la investigación ulterior en IA fue influenciada por este notable logro. Otros proyectos influyentes incluyen el Stanford Cart y el CMU Rover (Moravec, 1983). Cox y Wilfong (1990) describe trabajo clásico en vehículos autónomos.

El campo de la construcción de mapas en robótica ha evolucionado a partir de dos orígenes distintos. El primer hilo empezó con el trabajo de Smith y Cheeseman (1986), que aplicó los filtros de Kalman simultáneamente a los problemas de localización y problema de construcción de mapas. Este algoritmo fue implementado por primera vez por Moutarlier y Chatila (1989), y más tarde mejorado por Leonard y Durrant-Whyte (1992). Dissanayake *et al.* (2001) describe el estado del arte. El segundo hilo comenzó con el desarrollo de la representación de la **rejilla de ocupación** para la construcción probabilística de mapas, que especifica la probabilidad de que cada localización (x, y) esté ocupada por un obstáculo (Moravec y Elfes, 1985). Una revisión del estado del arte en la

REJILLA DE OCUPACIÓN

generación de mapas puede encontrarse en (Thrun, 2002). Kuipers y Levitt (1988) fueron los primeros en proponer un mapeado topológico en lugar de métrico, motivado por los modelos de cognición espacial humana.

Las primeras técnicas de localización del robot fueron examinadas por Borenstein *et al*. (1996). A pesar de que el filtrado de Kalman era bien conocido como método de localización en la teoría de control por décadas, la formulación probabilística general del problema de localización no apareció en la literatura sobre IA hasta mucho después, a través del trabajo de Tom Dean y sus compañeros (1990, 1990) y Simmons y Koenig (1995). El trabajo posterior introdujo el término de **localización de Markov**. La primera aplicación en el mundo real de esta técnica fue realizada por Burgard *et al*. (1999), mediante una serie de robots que fueron desplegados en museos. La localización Monte Carlo basada en filtros de partículas fue desarrollada por Fox *et al*. (1999) y ahora es ampliamente utilizada. El **filtro de partículas Rao-Blackwellizado** combina filtrado de partículas para la localización del robot con un filtrado exacto para la construcción del mapa (Murphy y Rusell, 2001; Montemerlo *et al*., 2002).

LOCALIZACIÓN DE MARKOV

FILTRADO DE PARTÍCULAS DE RAO-BLACKWELLIZED

La investigación en robótica móvil ha sido estimulada en la última década por dos competiciones importantes. La competición anual de robots móviles de la AAAI empezó en 1992. El primer ganador de la competición fue CARMEL (Congdon *et al*., 1992). El progreso ha sido estable e impresionante: en la competición más reciente (2002), los robots tenían que entrar dentro del complejo de la conferencia, encontrar el camino a la mesa de registro, registrarse para la conferencia, y dar una charla. La competición **Robocup**, iniciada en 1995 por Kitano y compañeros (1997), se ha marcado como objetivo para 2050 «desarrollar un equipo de robots humanoides totalmente autónomos que puedan ganar al campeón del mundo humano al fútbol». El juego se desarrolla en ligas para robots simulados, robots con ruedas de diferentes tamaños, y robots Sony Aibo de cuatro piernas. En 2002, el evento atrajo equipos de casi 30 países diferentes y unos 100.000 espectadores.

ROBOCUP

MÁQUINAS MANO-OJO

El estudio de robots manipuladores, llamados originalmente **máquinas mano-ojo**, ha evolucionado en líneas bastante diferenciadas. El primer gran logro en crear una máquina mano ojo fue el MH-1 de Heinrich Ernst, descrito en su tesis doctoral del MIT (Ernst, 1961). El proyecto Machine Intelligence de Edimburgo también evidenció un sistema de ensamblaje impresionante basado en visión llamado FREDDY (Michie, 1972). Después de estos esfuerzos pioneros, una gran parte del trabajo se centró en algoritmos geométricos para problemas de planificación de movimiento deterministas y enteramente observables. La complejidad PSPACE de la planificación del movimiento de un robot se mostró en un artículo de Reif (1979). La representación del espacio de configuración es debida a Lozano-Perez (1983). Altamente influyentes fueron una serie de artículos de Schwartz y Shafir sobre lo que ellos llamaron problemas de **motores de piano** (Schwartz *et al*., 1987).

MOTORES DE PIANO

La descomposición recursiva en celdas para la planificación del espacio de configuración fue introducida por Brooks y Lozano-Pérez (1985) y mejorada significativamente por Zhu y Latombe (1991). Los primeros algoritmos de esqueletización se basaron en los diagramas de Voronoi (Rowat, 1979) y los **grafos de visibilidad** (Wesley y Lozano-Pérez, 1979). Guibas *et al*. (1992) desarrollaron técnicas eficientes para calcular los diagramas de Voronoi incrementalmente, y Choset (1996) generalizó los diagramas

GRAFOS DE VISIBILIDAD

de Voronoi a problemas de planificación de movimiento mucho más amplios. La tesis de doctorado de John Canny (1988) estableció el primer algoritmo exponencial para la planificación de movimiento usando un método diferente de esqueletización llamado el algoritmo **silueta**. El texto de Jean-Claude Latombe (1991) cubre una variedad de aproximaciones al problema de planificación de movimiento. (Kavraki *et al*., 1996) desarrolló mapas de carretera probabilísticas, que son actualmente el método más efectivo. La planificación de movimiento precisa con sensorización limitada fue investigada por (Lozano-Pérez *et al*., 1984) y Canny y Reif (1987) usando la idea de intervalo de incertidumbre en lugar de la de incertidumbre probabilística. La navegación basada en señales (Lazanas y Latombe, 1992) usa algunas de estas mismas ideas en el campo de los robots móviles.

SILUETA

El control de robots como sistemas dinámicos (ya sea para manipulación o navegación) ha generado una inmensa cantidad de literatura, la cual este capítulo apenas toca. Los trabajos importantes incluyen una trilogía sobre control de impedancia por Hogan (1985) y un estudio general de la dinámica del robot por Featherstone (1987). Dean and Wellman (1991) se encuentran entre los primeros que intentaron enlazar la teoría de control y los sistemas de planificación de IA. Debemos tres libros clásicos sobre las matemáticas de la manipulación robótica a Paul (1981), Craig (1989) y Yoshikawa (1990). El área del **agarre** es también importante en robótica, el problema de determinar un agarre estable es bastante difícil (Mason y Salisbury, 1985). El agarre eficiente requiere sensorización de toque, o **retroalimentación táctil**, para determinar fuerzas de contacto y detectar deslizamiento (Fearing Hollerbach, 1985).

AGARRE

RETROALIMENTACIÓN TÁCTIL

El control por campo de potencial, que intenta resolver la planificación de movimiento y los problemas de control simultáneamente, fue introducido en la literatura sobre robótica por Khatib (1986). En la robótica móvil, esta idea fue vista como una solución práctica al problema de la evitación de obstáculos, y fue más tarde extendido en un algoritmo llamado **histogramas de campo vectorial** por Borenstein (1991). Las funciones de navegación, la versión robótica de la política de control para los MDPs deterministas, fueron introducidas por Koditschek (1987).

HISTOGRAMAS DE CAMPO VECTORIAL

El tópico de arquitecturas *software* para robótica genera mucho debate. El bien conocido candidato de la IA (la arquitectura de tres capas) proviene del diseño de Shakey y es revisado por Gat (1998). La arquitectura de subsumpción es debida a Rodney Brooks (1986), a pesar de que ideas similares fueron desarrolladas independientemente por Braitenberg (1984), cuyo libro, *Vehicles*, describe una serie de robots simples basados en la aproximación de comportamientos. El éxito del robot hexápodo de Brook fue seguido por unos cuantos proyectos. Connell, en su tesis (1989), desarrolló un robot móvil capaz de recoger objetos que era plenamente reactivo. Extensiones del paradigma basado en comportamientos en sistemas multi-robot pueden ser encontrados en (Mataric, 1997) y (Parker, 1996). GRL (Horswill, 2000) y COLBERT (Konolige, 1997) abstraen las ideas de la robótica basada en comportamientos concurrentes en lenguajes de control de robots. Arkin (1998) inspecciona el estado del arte.

Los **autómatas situados** (Rosenschein, 1985; Kaelbling y Rosenschein, 1990), descritos en el Capítulo 7, han sido también utilizados para el control de robots móviles en tareas de exploración y entrega. Los autómatas situados están estrechamente relacionados con diseños basados en comportamientos en el sentido en que consisten en autómatas

finitos que rastrean aspectos del estado del entorno usando circuitería combinatoria. Mientras que la aproximación basada en comportamientos acentúa la ausencia de representación explícita, los autómatas situados son construidos algorítmicamente a partir de modelos declarativos del entorno de tal forma que el contenido representacional de cada registro de estado está bien definido.

Existen varios buenos libros recientes sobre robótica móvil. Además de los libros mencionados anteriormente, la colección de Kortenkamp *et al*. (1998) provee una visión general de sistemas y arquitecturas de robots móviles contemporáneas. Dos textos recientes de Dudek y Jenkin (2000) y Murphy (2000) cubren la robótica de un modo más general. Un libro reciente sobre manipulación robótica escribe en la dirección de tópicos avanzados como el movimiento adaptado (Mason, 2001). La mayor conferencia sobre robótica es la *IEEE International Conference on Robotics and Automation*. Las publicaciones sobre robótica incluyen *IEEE Robotics and Automation*, el *International Journal of Robotics Research*, y *Robotic and Autonomous Systems*.

EJERCICIOS

25.1 La localización de Monte Carlo está *umbralizada* para cualquier tamaño de muestra finito (por ejemplo, el valor esperado de la localización computada por el algoritmo difiere del valor realmente esperado) debido a la forma en que trabaja el filtrado de partículas. En esta pregunta, se pide cuantificar este umbral.

Para simplificar, consideremos un mundo con cuatro posibles localizaciones para el robot: $X = \{x_1, x_2, x_3, x_4\}$. Inicialmente, dibujamos $N \geq 1$ muestras uniformemente entre estas localizaciones. Como es habitual, es perfectamente aceptable si más de una muestra se genera a partir de las localizaciones X. Sea Z la variable sensor booleana caracterizada por las siguientes probabilidades condicionales:

$$\begin{array}{ll} P(z \mid x_1) = 0{,}8 & P(\neg z \mid x_1) = 0{,}2 \\ P(z \mid x_2) = 0{,}4 & P(\neg z \mid x_2) = 0{,}6 \\ P(z \mid x_3) = 0{,}1 & P(\neg z \mid x_3) = 0{,}9 \\ P(z \mid x_4) = 0{,}1 & P(\neg z \mid x_4) = 0{,}9 \end{array}$$

MCL usa estas probabilidades para generar pesos de partículas, que son posteriormente normalizadas y usadas en el proceso de remuestreo. Por simplicidad, asumamos que sólo generamos una única nueva muestra en el proceso de remuestreo, independientemente de N. Esta muestra puede corresponder a alguna de las cuatro localizaciones en X. De esta forma, el proceso de muestreo define una distribución de probabilidad sobre X.

a) ¿Cuál es la distribución de probabilidad sobre X resultante para esta nueva muestra? Responda esta pregunta de forma separada para $N = 1,\ldots,10$ y para $N = \infty$.

b) La diferencia entre dos distribuciones de probabilidad P y Q puede ser medida por la divergencia KL, que se define como

$$KLP(P, Q) = \sum_i P(x_i)\, logP(x_i) \backslash Q(x_i)$$

¿Cuáles son las divergencias KL entre las distribuciones en (a) y la posterior verdadera?

c) ¿Qué modificación en la formulación del problema (¡no en el algoritmo!) garantizaría que el estimador específico anterior no está umbralizado incluso para valores finitos de N? Indique al menos dos de estas modificaciones (cada una de las cuales debe ser suficiente).

25.2 Implemente la localización de Monte Carlo para un robot simulado con sensores de rango. Un mapa de rejilla y datos de rango están disponibles en el repositorio de código de **aima.cs.berkeley.edu**. El ejercicio está completado si usted puede demostrar una localización global con éxito del robot.

25.3 Considere el brazo robótico mostrado en la Figura 25.12. Asuma que el elemento de la base del robot tiene 60cm de largo y que el brazo y el antebrazo miden cada uno 40cm de largo. Como se discute en el Apartado 25.4, la cinemática inversa de un robot habitualmente no es única. Enuncie una solución explícitamente cerrada para la cinemática inversa de este brazo. ¿Bajo qué condiciones es la solución única?

25.4 Implemente un algoritmo para calcular el diagrama de Voronoi de un entorno 2-D arbitrario, descrito por una matriz booleana $n \times n$. Ilustre su algoritmo dibujando el diagrama de Voronoi para 10 mapas interesantes. ¿Cuál es la complejidad de su algoritmo?

25.5 Este ejercicio explora la relación entre el espacio de configuración y el espacio de trabajo usando los ejemplos mostrados en la Figura 25.25.

a) Considere la configuración del robot mostrada de la Figura 25.25(a) a (c), ignorando el obstáculo mostrado en cada uno de los diagramas. Dibuje las configuraciones del brazo correspondientes en el espacio de configuración (*Pista*: cada configuración del brazo se mapea como un único punto en el espacio de configuración, como se ilustra en la Figura 25.12(b).)

b) Dibuje el espacio de configuración para cada uno de los diagramas de espacio de trabajo en Figura 25.25(a)-(c). (*Pista*: Los espacios de configuración comparten con el mostrado en la Figura 25.25(a) la región que corresponde a la autocolisión, pero las diferencias provienen de la falta de obstáculos circundantes y las diferentes localizaciones de los obstáculos en estas figuras individuales.)

c) Para cada uno de los puntos negros en la Figura 25.25(e)-(f), dibuje las correspondientes configuraciones del brazo robótico en el espacio de trabajo. Ignore las regiones sombreadas en este ejercicio.

d) Los espacios de configuración mostrados en la Figura 25.25(e)-(f) han sido todos generados por un único obstáculo en el espacio de trabajo (sombreado oscuro), más las limitaciones producidas por la auto-colisión (sombreado claro). Dibuje, para cada diagrama, el obstáculo del área de trabajo que se corresponde con el área oscura sombreada.

e) La Figura 25.25(d) ilustra que un obstáculo plano simple puede descomponer el área de trabajo en dos regiones desconectadas. ¿Cuál es el máximo número de regiones desconectadas que pueden ser creadas insertando un obstáculo plano en un área de trabajo conectada sin obstáculos, para un robot de dos grados de

Figura 25.25 Diagramas para el Ejercicio 25.5.

libertad? Dé un ejemplo, y razone por qué no se puede crear un número mayor de regiones desconectadas. ¿Qué ocurriría con un obstáculo no plano?

25.6 Considere el robot simplificado mostrado en la Figura 25.26. Suponga que las coordenadas cartesianas del robot se conocen siempre, como las de su localización objetivo. Sin embargo, las localizaciones de los obstáculos son desconocidas. El robot puede sensorizar obstáculos en su proximidad inmediata, como se ilustra en esta figura. Por simplicidad, asumimos que el movimiento del robot está libre de ruido, y el espacio de estados es discreto. La Figura 22.26 es sólo un ejemplo; en este ejercicio se requiere que indique todos los posibles mundos de rejilla con una ruta válida desde el comienzo hasta la localización objetivo.

a) Diseñe un controlador deliberativo que garantice que el robot siempre alcanza su localización objetivo cuando esto sea posible. El controlador deliberativo puede memorizar medidas en forma de un mapa que está siendo obtenido mientras el robot se mueve. Entre movimientos individuales, puede pasar cualquier tiempo arbitrario deliberando.

b) Ahora diseñe un controlador *reactivo* para la misma tarea. Este controlador no puede memorizar medidas anteriores de los sensores (¡no puede construir un mapa!). En lugar de ello, debe tomar todas sus decisiones basándose en medi-

Figura 25.26 Robot simplificado en un laberinto. *Véase* Ejercicio 25.6.

das actuales, que incluyen el conocimiento de su propia localización y la del objetivo. El tiempo para tomar una decisión debe ser independiente del tamaño del entorno o del número de pasos temporales anteriores. ¿Cuál es el número máximo de pasos que debe tomar el robot para llegar al objetivo?

c) ¿Cómo se comportarían sus controladores de (a) y (b) si se aplicara alguna de las siguientes seis condiciones?: espacio de estados continuo, ruido en la percepción, ruido en el movimiento, ruido en la percepción y en el movimiento, localización desconocida del objetivo (el objetivo sólo puede ser detectado cuando se encuentre en el interior del rango de los sensores), u obstáculos móviles. Para cada condición y para cada controlador, dé un ejemplo de una situación en la que el robot falle (o explique por qué no puede fallar).

25.7 En la Figura 25.22(b), encontrábamos un autómata finito aumentado para el control de una pierna del robot hexápodo. En este ejercicio, el objetivo es el diseño de un autómata finito aumentado que, cuando esté combinado con seis copias del controlador individual para una pierna, resulte en una locomoción eficiente y estable. Para este propósito, debe aumentar el controlador para la pierna para pasar mensajes a su nuevo autómata finito aumentado, y para esperar hasta que otros mensajes le lleguen. Razone por qué su controlador es eficiente, de tal forma que no gaste energía innecesariamente (por ejemplo, al deslizarse las piernas) y que propulse al robot a unas velocidades razonablemente altas. Pruebe que su controlador satisface la condición de estabilidad dada en el Apartado 25.2.

25.8 (Este ejercicio fue por primera vez diseñado por Michael Genesereth y Nils Nilsson. Puede ser realizado desde alumnos de primer curso hasta alumnos ya graduados.) Los humanos están tan adaptados a tareas básicas como tomar tazas o apilar bloques, que a menudo olvidan lo complejas que son estas tareas. En este ejercicio descubrirá la complejidad y recapitulará los últimos 30 años de desarrollos en robótica. Primero, elija una tarea, como construir un arco con tres bloques. Entonces, construya un robot a partir de cuatro humanos de la siguiente forma:

Cerebro. El trabajo del cerebro es obtener un plan para conseguir el objetivo, y dirigir las manos en la ejecución del plan. El cerebro recibe entrada de los ojos, pero *no puede ver la escena directamente*. El cerebro es el único que conoce cuál es el objetivo.

Ojos. El trabajo de los ojos es dar una breve descripción de la escena al cerebro. Los ojos deben permanecer unos cuantos pies alejados del entorno de trabajo, y pueden proporcionar descripciones cualitativas (como «Hay una caja roja que se encuentra encima de una caja verde, que está a su lado») o descripciones cuantitativas («La caja verde está aproximadamente dos pies a la izquierda del cilindro azul»). Los ojos también pueden responder a preguntas del cerebro, como «¿hay algún hueco entre la mano izquierda y la caja roja?». Si tiene una cámara de vídeo, apunte a la escena y permita a los ojos mirar al visor de la cámara, pero no directamente a la escena.

Mano izquierda y **mano derecha**. Una persona ejecuta el papel de cada mano. Las dos manos permanecen cerca una de la otra; la mano izquierda utiliza sólo su mano izquierda, y la mano derecha utiliza sólo su mano derecha. Las manos ejecutan sólo comandos simples del cerebro, por ejemplo, «mano izquierda, muévete dos pulgadas hacia delante». No pueden ejecutar otros comandos que no sean movimientos; por ejemplo, «toma la caja» no es algo que la mano pueda hacer. Para evitar las trampas, puede dejar a las manos llevar guantes, o que operen pinzas. Las manos deben *tener los ojos vendados*. La única capacidad de sensorización que tienen es la habilidad de decir cuándo su ruta es bloqueada por un objeto inamovible como una mesa o la otra mano. En estos casos, pueden emitir un sonido para informar al cerebro de esta dificultad.

26 Fundamentos filosóficos

En este capítulo examinamos lo que significa pensar y si los artefactos podrían y deberían alguna vez llegar a hacerlo.

Como se mencionó en el Capítulo 1, los filosófos existían mucho antes que los computadores y llevaban tiempo intentando solucionar algunas cuestiones relacionadas con la IA: ¿cómo trabaja la mente? ¿Es posible que las máquinas actúen de forma inteligente, igual que las personas? Y si así fuera, ¿tendrían mentes? ¿Cuáles son las implicaciones éticas de las máquinas inteligentes? A lo largo de los primeros 25 capítulos de este libro, hemos estudiado cuestiones de IA en sí misma, pero en este capitulo estudiaremos la agenda del filósofo.

HIPÓTESIS DE LA IA DÉBIL

HIPÓTESIS DE LA IA FUERTE

En primer lugar, observemos la terminología: los filósofos definen la **hipótesis de la IA débil** como la afirmación de que es posible que las máquinas actúen con inteligencia (o quizá mejor, *como si* fueran inteligentes); de la misma manera, la **hipótesis de la IA fuerte** consiste en la afirmación de que las máquinas sí piensan *realmente* (opuesto al pensamiento *simulado*).

La mayoría de los investigadores de IA dan por sentado la hipótesis de la IA débil, y no se preocupan por la hipótesis de la IA fuerte, con tal de que funcione su programa no les interesa si se llama simulación de inteligencia o inteligencia real. Sin embargo, todos deberían preocuparse por las implicaciones éticas de su trabajo.

26.1 IA débil: ¿pueden las máquinas actuar con inteligencia?

Algunos filósofos han intentado demostrar que la IA es imposible; que las máquinas no tendrán la posibilidad de actuar inteligentemente. Algunos han utilizado argumentos que tratan de dar el alto a la investigación en IA:

> La Inteligencia Artificial *abordada desde dentro del culto al computacionalismo* no tendrá ni siquiera un atisbo de fantasma de posibilidad de producir resultados duraderos... Es hora de desviar los esfuerzos de los investigadores en IA, y la gran cantidad de dinero disponible para su soporte, y dirigirse a caminos distintos del enfoque computacional (Sayre, 1993).

Obviamente, si la IA es imposible o no lo es, dependerá de cómo se defina. En esencia, la IA consiste en la búsqueda del mejor programa agente en una arquitectura dada. Con esta formulación, la IA es posible por definición: para cualquier arquitectura digital de k bits de almacenamiento existirán exactamente 2^k programas agente y todo lo que habrá que hacer para encontrar el mejor es enumerarlos y probar todos ellos. Esto podría no ser viable para una k grande, pero los filósofos abordan más la teoría que la práctica.

Nuestra definición de IA funciona bien para el problema de encontrar un buen agente, dependiendo de la arquitectura. Por tanto, nos sentimos tentados a acabar esta sección aquí mismo, respondiendo afirmativamente a la pregunta formulada en el título. Sin embargo, los filósofos están interesados en el problema de comparar dos arquitecturas, la humana y la de la máquina. Además, ellos por tradición han formulado la pregunta de la siguiente manera: **«¿Pueden pensar las máquinas?»** Desgraciadamente, esta cuestión no está bien definida. Para ver por qué, consideremos las dos cuestiones siguientes:

¿PUEDEN PENSAR LAS MÁQUINAS?

- ¿Pueden volar las máquinas?
- ¿Pueden nadar las máquinas?

La mayoría de las personas están de acuerdo en que la respuesta a la primera cuestión es sí, que los aviones pueden volar, pero la respuesta a la segunda es no; los barcos y los submarinos se mueven por el agua, pero eso no es nadar. Sin embargo, ni las preguntas ni sus respuestas afectan en absoluto a las vidas laborales de los ingenieros aeronáuticos ni navales, ni a las de los usuarios de sus productos. Las respuestas no tienen mucho que ver con el diseño o con las características de los aviones o de los submarinos, y sin embargo sí tienen que ver mucho más con la forma en que se han elegido utilizar las palabras. La palabra nadar («swim» en inglés) ha llegado a tener el significado de «moverse por el agua mediante el movimiento de las partes del cuerpo», mientras que la palabra «fly» (volar) no tiene dicha limitación en un medio de locomoción[1]. La posibilidad práctica de las «máquinas pensantes» lleva viviendo con nosotros durante sólo 50 años o así, tiempo insuficiente para que los angloparlantes se decidan a dar un significado a la palabra «pensar».

Alan Turing, en su famoso artículo «Computing Machinery and Intelligence» (Turing, 1950), sugirió que en vez de preguntar si las máquinas pueden pensar, deberíamos preguntar si las máquinas pueden aprobar un test de inteligencia conductiva (de comportamiento), conocido como el Test de Turing. La prueba se realiza para que el programa mantenga una conversación durante cinco minutos (mediante mensajes escritos en línea, *online*) con un interrogador (interlocutor). Éste tiene que averiguar si la conversación se está llevando a cabo con un programa o con una persona; si el programa engaña al interlocutor un 30 por ciento del tiempo, este pasará la prueba. Turing conjeturó que, hacia

[1] En ruso, el equivalente de «nadar» *sí* se aplica a los barcos.

el año 2000, un computador con un almacenamiento de 10^9 unidades podría llegar a programarse lo suficientemente bien como para pasar esta prueba, pero no estaba en lo cierto. Algunas personas *han* sido engañadas durante cinco minutos; por ejemplo, el programa ELIZA y el chatbot en Internet llamado MGONZ han engañado a personas ignorantes que no se daban cuenta de que estaban hablando con un programa; el programa ALICE engañó a un juez en la competición del Loebner Prize en el año 2001. Sin embargo, ningún programa se ha acercado al criterio del 30 por ciento frente a jueces con conocimiento, y el campo en su conjunto de la IA no ha prestado mucha atención a los tests de Turing.

Turing también examinó una gran gama de posibles objeciones ante la posibilidad de las máquinas inteligentes, incluyendo virtualmente aquellas que han aparecido medio siglo después de que apareciera este artículo. Examinaremos algunas de ellas.

El argumento de incapacidad

El «argumento de incapacidad» afirma que «una máquina nunca puede hacer *X*». Como ejemplos de *X*, Turing enumera las siguientes acciones:

> Ser amable, tener recursos, ser guapo, simpático, tener iniciativas, tener sentido del humor, distinguir lo correcto de lo erróneo, cometer errores, enamorarse, disfrutar con las fresas con nata, hacer que otra persona también se enamore, aprender de la experiencia, utilizar palabras de forma adecuada, ser el tema de su propio pensamiento, tener tanta diversidad de comportamientos como el hombre, hacer algo realmente nuevo.

Turing tuvo que utilizar su intuición para adivinar aquello que en un futuro sería posible, pero nosotros tenemos el privilegio de poder mirar hacia atrás y ver qué es lo que ya pueden hacer los computadores. Es innegable que los computadores actualmente hacen muchas cosas que anteriormente eran sólo del dominio humano. Los programas juegan a la ajedrez, a las damas y a otros juegos, inspeccionan piezas de las líneas de producción, comprueban la ortografía en los documentos de los procesadores de texto, conducen coches y helicópteros, diagnostican enfermedades, y hacen otros cientos de tareas tan bien o mejor que los hombres. Los computadores han hecho pequeños pero significativos descubrimientos, en Astronomía, Matemáticas, Química, Mineralogía, Biología, Informática y otros campos que necesitan rendimiento a nivel de experto.

Debido a lo que conocemos actualmente acerca de los computadores, no es sorprendente que sean también buenas en problemas combinatorios tales como los del juego del ajedrez. Sin embargo, los algoritmos también funcionan a nivel humano en tareas que aparentemente se relacionan con el juicio humano, o como apunta Turing, «aprender a partir de la experiencia» y la capacidad de «distinguir lo que es correcto de lo incorrecto». Ya en el año 1955, Paul Meehl (*véase* también Grove y Meehl, 1996) estudió los procesos de la toma de decisiones de expertos formados en tareas subjetivas como predecir el éxito de un alumno en un programa de formación, o la reincidencia de un delincuente. De 20 estudios que Meehl examinó, en 19 de ellos encontró que sencillos algoritmos de aprendizaje estadístico (tal como la regresión lineal y Bayes simple) predicen mejor que los expertos. Desde el año 1999, el Educational Testing Service (Servicio de Exámenes Educativo) ha utilizado un programa automatizado para calificar millones de

preguntas de redacciones en el examen GMAT. Este programa concuerda con los examinadores en un 97 por ciento, aproximadamente al mismo nivel de concordancia entre dos personas (Burstein *et al.,* 2001).

Es evidente que los computadores pueden hacer muchas cosas tan bien o mejor que el ser humano, incluso cosas que las personas creen que requieren mucha intuición y entendimiento humano. Por supuesto, esto no significa que los computadores utilicen la intuición y el entendimiento para realizar estas tareas, las cuales no forman parte del *comportamiento*, y afrontamos dichas cuestiones en otro sitio, sino que la cuestión es que la primera conjetura sobre los procesos mentales que se requieren para producir un comportamiento dado suele ser equivocada. También es cierto, desde luego, que existen todavía muchas tareas en donde los computadores no sobresalen (por no decirlo más bruscamente), incluida la tarea de Turing de mantener una conversación abierta.

La objeción matemática

Es bien conocido, a través de los trabajos de Turing (1936) y Gödel (1931), que ciertas cuestiones matemáticas, en principio, no pueden ser respondidas por sistemas formales concretos. El teorema de la incompletitud de Gödel (*véase* el Apartado 9.5) es el ejemplo más conocido en este respecto. En resumen, para cualquier sistema axiomático formal F lo suficientemente potente como para hacer aritmética, es posible construir una «sentencia Gödel» $G(F)$ con las propiedades siguientes:

- G(F) es una sentencia de F, pero no se puede probar dentro de F.
- Si F es consistente, entonces $G(F)$ es verdadero.

Filósofos como J. R. Lucas (1961) han afirmado que este teorema demuestra que las máquinas son mentalmente inferiores a los hombres, porque las máquinas son sistemas formales limitados por el teorema de la incompletitud, es decir no pueden establecer la verdad de su propia sentencia Gödel, mientras que los hombres no tienen dicha limitación. Esta afirmación ha provocado mucha controversia durante décadas, generando muchos libros entre los que se incluyen dos libros del matemático Sir Roger Penrose (1989, 1994) quien repite esta afirmación con nuevos giros (como por ejemplo, la hipótesis de que los hombres son diferentes porque sus cerebros operan por la gravedad cuántica). Examinemos solamente tres de los problemas de esta afirmación.

En primer lugar, el teorema de la incompletitud de Gödel se aplica sólo a sistemas formales que son lo suficientemente potentes como para realizar aritmética. Aquí se incluyen las máquinas Turing, y la afirmación de Lucas en parte se basa en la afirmación de que los computadores son máquinas de Turing. Esta es una buena aproximación, pero no es del todo verdadera. Aunque los computadores son finitos, las máquinas de Turing son infinitas, y cualquier computador por tanto se puede describir como un sistema (muy grande) en la lógica proposicional, la cual no está sujeta al teorema de incompletitud de Gödel.

En segundo lugar, un agente no debería avergonzarse de no poder establecer la verdad de una sentencia aunque otros agentes sí puedan. Consideremos la sentencia siguiente

J. R. Lucas no puede consecuentemente afirmar que esta sentencia es verdadera.

Si Lucas afirmara esta sentencia, entonces se estaría contradiciendo a sí mismo, por tanto Lucas no puede afirmarla consistentemente, y de aquí que esta sentencia sea verdadera. (La sentencia no puede ser falsa, porque si lo fuera Lucas entonces no podría afirmarla consecuentemente, por tanto sería verdadera.) Así pues, hemos demostrado que existe una sentencia que Lucas no puede afirmar consecuentemente mientras que otras personas (y máquinas) sí pueden. Sin embargo, esto no hace que cambiemos de idea respecto a Lucas. Por dar otro ejemplo, ninguna persona podría calcular la suma de 10 billones de números de 10 dígitos en su vida, en cambio un computador podría hacerlo en segundos. Sin embargo, no vemos esto como una limitación fundamental en la habilidad de pensar del hombre. Durante miles de años los hombres se han comportado de forma inteligente antes de que se inventaran las máquinas, de manera que no es improbable que el razonamiento matemático no tenga más que una función secundaria en lo que implica ser inteligente.

En tercer lugar, y de manera mucho más importante, aunque reconozcamos que los computadores tienen limitaciones sobre lo que pueden demostrar, no existen evidencias de que los hombres sean inmunes ante esas limitaciones. Es realmente sencillo demostrar con rigor que un sistema formal no puede hacer *X*, y afirmar entonces que los hombres *pueden* hacer *X* utilizando sus propios métodos informales, sin dar ninguna evidencia de esta afirmación. En efecto, es imposible demostrar que los hombres no están sujetos al teorema de incompletitud de Gödel, porque cualquier prueba rigurosa contendría una formalización del talento humano declarado como no formalizable. De manera que nos quedamos con el llamamiento a la intuición de que los hombres, de alguna forma, pueden realizar hazañas superhumanas de comprensión matemática. Esta atracción se expresa con argumentos como «debemos asumir nuestra propia consistencia, si el pensamiento puede ser posible» (Lucas, 1976). Sin embargo ciertamente se sabe que los hombres son inconsistentes. Esto es absolutamente verdadero para el razonamiento diario, pero también es verdadero para un pensamiento matemático cuidadoso. Un ejemplo muy conocido es el problema del mapa de cuatro colores. En 1879, Alfred Kempe publicó una prueba que tuvo una gran acogida y contribuyó a que le eligieran Fellow de Royal Society. Sin embargo, en 1890, Percy Heawood apuntó que existía un error y el teorema quedó sin demostrar hasta el año 1977.

El argumento de la informalidad

Una de las críticas más persistentes e influyentes de la IA como empresa la realizó Turing mediante su «argumento de la informalidad del comportamiento». En esencia, esta afirmación consiste en que el comportamiento humano es demasiado complejo para poder captarse mediante un simple juego de reglas y que debido a que los computadores no pueden nada más que seguir un conjunto (juego) de reglas, no pueden generar un comportamiento tan inteligente como el de los hombres. En IA la incapacidad de capturarlo todo en un conjunto de reglas lógicas se denomina **problema de cualificación** (*véase* Capítulo10).

El filósofo que ha propuesto principalmente este punto de vista ha sido Hubert Dreyfus, quien elaboró una serie de críticas influyentes a la Inteligencia Artificial: *¿Qué*

es lo que no pueden hacer los computadores? (1972), *¿Qué es lo no pueden hacer todavía los computadores?* (1992). Junto con su hermano elaboró también *Mind Over Machine* (1986).

La postura que critican se vino a llamar «Good Old-Fashioned AI» (IA muy anticuada), GOFAI, término que empezó a utilizar Haugeland (1985). Se supone que este término afirma que todo comportamiento inteligente puede ser capturado por un sistema que razona lógicamente a partir de un conjunto de hechos y reglas, los cuales describen el dominio. Por tanto, se corresponde con el agente lógico más simple que se describió en el Capítulo7. Dreyfus está en lo cierto cuando dice que los agentes son vulnerables al problema de la cualificación. Como se vio en el Capítulo 13, los sistemas de razonamiento probabilístico son más adecuados para dominios abiertos. La crítica de Dreyfus por lo tanto no va en contra de los computadores *per se*, sino en contra de una forma en particular de programarlos. Sin embargo, sería razonable suponer que un libro llamado *Lo que no pueden hacer los sistemas lógicos de primer orden basados en reglas sin aprender* podría haber tenido menos impacto.

Bajo el punto de vista de Dreyfus, la pericia del hombre incluye el conocimiento de algunas reglas, pero solamente como un «contexto holístico» o «conocimiento base» *(background)* dentro del que operan los hombres. Proporciona como ejemplo el comportamiento social adecuado al dar o recibir regalos: «Normalmente se responde simplemente en las circunstancias adecuadas y dando el regalo adecuado». Al parecer hay que «tener un sentido directo de cómo hay que hacer las cosas y qué esperar». Esta misma afirmación se realiza en el contexto del juego del ajedrez: «Un maestro de ajedrez tendría que averiguar simplemente qué hacer, pero un buen maestro simplemente observa el tablero como exigiendo un cierto movimiento... y obtiene la respuesta apropiada rápidamente en su cabeza». Es cierto que gran parte de los procesos del pensamiento de una persona que da un regalo o de un gran maestro en ajedrez se llevan a cabo a un nivel que no está abierto a la introspección por la mente consciente. Sin embargo, esto no significa que no existan los procesos de pensamiento. Una cuestión importante que Dreyfus no responde es *cómo* aparece el movimiento de ajedrez adecuado en la cabeza del gran maestro. Esto nos lleva a pensar en un comentario de Dennett (1984):

> Es como si los filósofos se fueran a proclamar expertos en explicar los métodos de los magos en el escenario, y entonces cuando preguntamos cómo hace el truco del serrucho para partir en dos a una mujer, ellos dan la explicación de que es totalmente evidente: el mago no parte en dos a la mujer con la sierra, simplemente parece que lo hace. «Pero, ¿Cómo lo hace?», y los filósofos responden «No es de nuestra incumbencia».

Dreyfus y Dreyfus (1986) proponen un proceso de adquisición de pericia en cinco etapas, comenzando con un procesamiento basado en reglas (del tipo propuesto en GOFAI) y terminando con la habilidad de seleccionar las respuestas correctas instantáneamente. Al realizar esta propuesta, Dreyfus y Dreyfus pasan en efecto de ser críticos a la IA a ser teóricos de IA, ya que proponen una arquitectura de redes neurales (neuronales) organizadas en una biblioteca de casos extensa, pero señalan algunos problemas. Afortunadamente, se han abordado todos sus problemas, algunos con éxito parcial y otros con éxito total. Entre estos problemas se incluyen los siguientes:

1. No se puede lograr una generalización buena de ejemplos sin un conocimiento básico. Afirman que no se sabe cómo incorporar el conocimiento básico en el proceso de aprendizaje de las redes neuronales. De hecho, en el Capítulo 19 vimos que existen técnicas para utilizar el conocimiento anterior en los algoritmos de aprendizaje. Sin embargo, esas técnicas dependen de la disponibilidad previa de conocimiento de forma explícita en los algoritmos de aprendizaje, algo que Dreyfus y Dreyfus niegan vigorosamente. Bajo nuestro punto de vista, esta es una buena razón para realizar un rediseño serio de los modelos actuales del procesamiento neuronal de forma que *puedan* sacar provecho del conocimiento aprendido anteriormente como lo hacen otros algoritmos de aprendizaje.
2. El aprendizaje de redes neuronales es una forma de aprendizaje supervisado (*véase* Capítulo18), que requiere la identificación anterior de las entradas relevantes y las salidas correctas. Por tanto, afirman que no puede funcionar autónomamente sin la ayuda de un entrenador humano. De hecho, el aprendizaje sin un profesor se puede conseguir mediante un **aprendizaje no supervisado** (Capítulo 20) y un **aprendizaje de refuerzo** (Capítulo 21).
3. Los algoritmos de aprendizaje no funcionan bien con muchas funciones, si seleccionamos un subgrupo de éstas, «no existe una forma conocida de añadir funciones nuevas, si el conjunto actual demuestra ser inadecuado para tener en cuenta los hechos aprendidos». De hecho, métodos nuevos tales como las máquinas vectoriales de soporte utilizan muy bien conjuntos grandes de funciones. Como vimos en el Capítulo 19, también existen formas importantes de generar funciones nuevas, aunque requiera más trabajo.
4. El cerebro es capaz de dirigir sus sensores para buscar la información relevante y procesarla para extraer aspectos relevantes para la situación actual. Sin embargo, afirman que «Actualmente, los detalles de este mecanismo ni se entienden y ni siquiera se hipotetizan para guiar la investigación en la IA». De hecho, el campo de la visión activa, respaldado por la teoría del valor de la información (Capítulo 16) tiene que ver exactamente con el problema de dirigir los sensores, y algunos robots ya han incorporado los resultados teóricos obtenidos.

En resumen, muchos de los temas que ha tratado Dreyfus, el conocimiento del sentido común básico, el problema de la cualificación, la incertidumbre, aprendizaje, formas compiladas de la toma de decisiones, la importancia de considerar agentes situados y no motores de interferencia incorpóreos, por ahora se han incorporado en el diseño estándar de agentes inteligentes. Bajo nuestro punto de vista, esta es una evidencia del progreso de la IA, y no de su imposibilidad.

26.2 IA fuerte: ¿pueden las máquinas pensar de verdad?

Muchos filósofos han afirmado que una máquina que pasa el Test de Turing no quiere decir que esté *realmente* pensando, sería solamente una *simulación* de la acción de

pensar. De nuevo esta objeción fue prevista por Turing, y cita unas palabras del Profesor Geoffrey Jefferson (1949):

> Hasta que una máquina pueda escribir un soneto o componer un concierto porque sienta los pensamientos y las emociones, y no porque haya una lluvia de símbolos, podría reconocer que la máquina iguala al cerebro, es decir, no sólo escribirlo sino que sepa que lo ha hecho.

Esto es lo que Turing llama el argumento de la **consciencia**, la máquina tiene que ser consciente de sus propias acciones y estados mentales. Aunque la consciencia sea un tema importante, el punto de vista clave de Jefferson se relaciona realmente con la **fenomenología**, o el estudio de la experiencia directa, es decir, la máquina tiene que sentir emociones realmente. Otros se centran en la **intencionalidad**, esto es, en la cuestión de si las creencias, deseos y otras representaciones supuestas de la máquina son de verdad algo que pertenece al mundo real.

La respuesta de Turing a esta objeción es interesante. Podría haber presentado razones para demostrar que las máquinas pueden de hecho ser conscientes (o tener fenomenología, o tener intenciones). En cambio, Turing mantiene que la cuestión no está bien definida al decir, «¿Pueden pensar las máquinas?» Además, por qué deberíamos insistir en un estándar más alto para las máquinas que el usado para los humanos. Después de todo, en la vida ordinaria no tenemos nunca una evidencia directa sobre los estados mentales internos de otras personas. No obstante, Turing dice que «En vez de argumentar constantemente sobre este punto de vista, es usual mantener la **convención educada** de que todos pensamos».

CONVENCIÓN EDUCADA

Turing argumenta que Jefferson estaría dispuesto a ampliar la convención educada a las maquinas si al menos tuviera experiencia con las que actúan de forma inteligente, y cita el diálogo siguiente el cual ha llegado a formar una parte muy significativa de la tradición oral de la IA y que debemos incluir:

> HOMBRE: En la primera línea de tu soneto que dice «te compararé con un día de verano», ¿no sería mejor decir «un día de primavera»?
> MÁQUINA: No rimaría.
> HOMBRE: Y qué tal un «día de invierno». Este sí qué rimaría bien.
> MÁQUINA: Sí, pero nadie quiere una comparación con un día de invierno.
> HOMBRE: ¿Podrías decir que Mr. Pickwick te recuerda a las navidades?
> MÁQUINA: En cierto modo, sí.
> HOMBRE: Sin embargo, las navidades representan un día de invierno, y no creo que a Mr. Pickwick le importara la comparación.
> MÁQUINA: No, creo que estés hablando en serio. Por un día de invierno se entiende un típico día de invierno, y no un día especial de Navidad.

Turing reconoce que la cuestión de la conciencia (consciencia) es difícil, pero niega que sea relevante para la práctica de la IA: «No quiero dar la impresión de que pienso que no hay misterio en torno a la conciencia… Sin embargo no creo que estos misterios tengan necesariamente que resolverse antes de la respuesta a la cuestión que estamos tratando en este trabajo». Coincidimos con Turing en que nos interesa crear programas que se comporten de forma inteligente y no en si alguien los declara reales o simulados. Por otro lado, muchos filósofos están especialmente interesados en esta cuestión. Como

ayuda para entenderlo tendremos en cuenta la cuestión de si otros artefactos se consideran reales.

En 1848, Frederick Wöhler sintetizó urea artificial por primera vez. Este fue un hecho importante porque probó que la química orgánica y la inorgánica se podían unir, cuestión discutida muy fuertemente. Una vez que se consiguió la síntesis, los químicos reconocieron que la urea artificial *era* urea, porque tenía todas las propiedades físicas adecuadas. Igualmente, los edulcorantes artificiales son innegablemente edulcorantes, y la inseminación artificial (la otra IA) es innegablemente inseminación. Por otro lado, las flores artificiales no son flores, y Daniel Dennett señala que el vino artificial Chateau Latour no sería vino Chateau Latour, aunque no se pudiera distinguir químicamente, simplemente porque no se fabricó en el lugar adecuado ni de la forma adecuada. Ni tampoco un Picasso artificial sería un Picasso, independientemente del aspecto que tenga el cuadro.

Podemos concluir diciendo que en algunos casos el comportamiento de un artefacto es importante, aunque en otros sea el pedigrí del artefacto lo que importa. Lo importante en cada caso parece ser una cuestión de convención. Sin embargo para las mentes artificiales, no existe una convención, y tenemos que depender de las intuiciones. El filósofo John Searle (1980) tiene una convención muy fuerte:

> Nadie piensa que la simulación por computador de una tormenta nos va a mojar… Y, ¿cual es la razón de que cualquier persona en su sano juicio suponga que la simulación por computador de los procesos mentales tendrían realmente procesos mentales? (pp. 37-38.)

Aunque sea fácil reconocer que las simulaciones por computador de las tormentas no nos van a mojar, no está claro cómo aplicar esta analogía a las simulaciones por computador de los procesos mentales. Después de todo, la simulación de una tormenta en Hollywood utilizando aspersores y máquinas de viento *sí* que moja a los actores. La mayoría de las personas se sienten cómodas diciendo que la simulación por computador de la suma es la suma, y la de un juego de ajedrez un juego de ajedrez. ¿Son los procesos mentales parecidos a las tormentas, o son más parecidos a la suma o al ajedrez, o se parecen al Chateau Latour y al Picasso..., o a la urea? Todo depende de la teoría de los estados y los procesos mentales.

FUNCIONALISMO

La teoría del **funcionalismo** dice que un estado mental es cualquier condición causal inmediata entre la entrada y la salida. Bajo la teoría funcionalista, dos sistemas con procesos causales isomórficos tendrían los mismos estados mentales. Por tanto, un programa informático podría tener los mismos estados mentales que una persona. Desde luego, todavía no hemos dicho lo que significa realmente «isomórficos», pero la suposición es que existe algún nivel de abstracción por debajo del cual no importa una implementación específica; siempre que los procesos sean isomórficos hasta este nivel, tendrán lugar los mismos estados mentales.

NATURALISMO BIOLÓGICO

En contraste, la teoría del **naturalismo biológico** dice que los estados mentales son características emergentes de alto nivel originadas por procesos neurológicos de bajo nivel *en las neuronas*, y lo que importa son las propiedades (no especificadas) de las neuronas. Así pues, los estados mentales no se pueden duplicar justo en la base de algún programa que tiene la misma estructura funcional con el mismo comportamiento de entrada y salida; necesitaríamos que el programa se ejecutara en una arquitectura con la

misma potencia causal que las neuronas. La teoría no dice por qué las neuronas tienen esta potencia causal, ni tampoco qué otras instanciaciones físicas podrían tenerla o no.

Para investigar estos dos puntos de vista examinaremos uno de los problemas más antiguos de la filosofía de la mente, y retomaremos tres experimentos pensados.

El problema de mente-cuerpo

PROBLEMA MENTE-CUERPO

El **problema mente-cuerpo** cuestiona cómo se relacionan los estados y los procesos mentales con los estados y los procesos (específicamente del cerebro) del cuerpo. Como si no fuera bastante difícil, vamos a generalizar el problema como un problema de «arquitectura-mente», para que nos permita hablar sobre la posibilidad de que las máquinas tengan mentes.

¿Por qué es un problema el problema mente-cuerpo? Para la primera dificultad nos remontaremos a René Descartes, quien abordó el tema de cómo un alma inmortal interactúa con un cuerpo mortal, y concluyó diciendo que el alma y el cuerpo son dos tipos de cosas diferentes, una teoría **dualista**. La teoría **monista**, frecuentemente llamada **materialismo**, mantiene que no existen cosas tales como almas inmateriales, sino sólo objetos materiales. Como consecuencia, los estados mentales, tales como sentir dolor, saber que alguien está montando a caballo, o creer que la capital de Austria es Viena, son estados del cerebro. John Searle, de forma concisa, resume esta idea con el slogan, «*Los cerebros producen las mentes*».

DUALISTA

MONISTA

MATERIALISMO

El materialista se debe enfrentar por lo menos con dos obstáculos serios. El primer problema es el de la **libertad de elección**: ¿Cómo puede ser que una mente puramente física, cuyas transformaciones están regidas por las leyes de la física, conserve todavía el libre albedrío? La mayoría de los filósofos consideran que este problema necesita una reconstitución cuidadosa de nuestra noción ingenua de libertad de elección, en vez de presentar una amenaza para el materialismo. El segundo problema tiene que ver con el tema general de la **conciencia** (y cuestiones de **entendimiento** y de **autoconocimiento** relacionadas, aunque no idénticas). Para simplificar, por qué se siente algo cuando se tienen ciertos estados cerebrales, mientras que probablemente no se siente nada al tener otros estados físicos (por ejemplo, al ser una roca).

LIBERTAD DE ELECCIÓN

CONCIENCIA

Para empezar a responder estas cuestiones, necesitamos formas de hablar sobre los estados del cerebro a niveles más abstractos que las configuraciones específicas de todos los átomos del cerebro de una persona en particular en un momento concreto. Por ejemplo, cuando pienso en la capital de Austria, mi cerebro sufre miríadas de cambios diminutos de un picosegundo al otro, pero estos no constituyen un cambio *cualitativo* en el estado del cerebro. Para tenerlo en cuenta, necesitamos una noción de *tipos* de estados del cerebro bajo la cual podamos juzgar si dos estados del cerebro pertenecen al mismo tipo o a un tipo diferente. Existen diferentes opiniones sobre lo que quiere decir *tipo* en este caso. Casi todo el mundo cree que si tomamos un cerebro y sustituimos los átomos de carbono por un nuevo conjunto de átomos de carbono[2], el estado mental no

[2] Quizás incluso los átomos de un isótopo de carbón diferente, como se hace algunas veces en los experimentos de exploración cerebral.

se verá afectado. Esto es bueno porque los cerebros reales están sustituyendo continuamente sus átomos mediante procesos metabólicos, y sin embargo esto en sí no parece causar perturbaciones mentales importantes.

ESTADOS INTENCIONALES

Ahora vamos a examinar una clase en particular de estado mental: las **actitudes proposicionales** (tratadas por primera vez en el Capítulo 10), conocidas también como **estados intencionales**. Estos son estados tales como creer, conocer, desear, temer, y otros más que se relacionan con algunos aspectos del mundo exterior. Por ejemplo, la creencia de que Viena es la capital de Austria es una creencia *sobre* una ciudad en particular y su estado. Nos vamos a cuestionar si es posible que los computadores tengan estados intencionales que nos ayuden a entender cómo caracterizar dichos estados. Por ejemplo, se podría decir que el estado mental de desear una hamburguesa difiere del estado de desear una pizza porque en el mundo real una hamburguesa y una pizza son cosas diferentes. Es decir, los estados intencionales tienen una conexión necesaria con otros objetos del mundo externo. Por otro lado, anteriormente, hemos argumentado que los estados mentales son estados del cerebro, y de aquí que los estados mentales de identidad o no-identidad se deberían determinar permaneciendo completamente «dentro de la cabeza», sin hacer referencia al mundo real. Para examinar este dilema retomaremos el experimento del pensamiento que intenta separar a los estados intencionales de sus objetos externos.

El experimento del «cerebro en una cubeta»

Imagínese que al nacer le extraen el cerebro de su cuerpo y lo colocan en una cubeta con una ingeniería maravillosa. Esta cubeta mantiene su cerebro y le permite crecer y desarrollarse. Al mismo tiempo, su cerebro recibe unas señales electrónicas de un simulador informático que pertenece a un mundo totalmente ficticio, y las señales motoras de su cerebro son interceptadas y utilizadas para modificar la simulación cuando sea adecuado[3]. A continuación, el estado del cerebro podría tener el estado mental *MueroPor (Yo, Hamburguesa*) aunque no tenga un cuerpo con el que sentir hambre ni sentido del gusto para experimentarlo, y puede que tampoco haya hamburguesas en el mundo real. En ese caso, ¿sería el mismo estado mental que el del cerebro en un cuerpo?

CONTENIDO EXTENSO

CONTENIDO ESTRECHO

Una forma de resolver el dilema es decir que el contenido de los estados mentales puede ser interpretado desde dos puntos de vista diferentes. La visión del «**contenido extenso**» interpreta el dilema desde el punto de vista de un observador omnisciente desde fuera con acceso a la situación completa, y que puede distinguir las diferencias del mundo. De esta manera, bajo el contenido extenso, las ideas del cerebro en una cubeta son diferentes de las de una persona «normal». El **contenido estrecho** sólo tiene en cuenta el punto de vista subjetivo interno, y bajo este punto de vista todas las creencias serían las mismas.

QUALIA

El pensamiento de que una hamburguesa es maravillosa tiene una cierta naturaleza intrínseca, esto es que existe algo que es como tener esa creencia. Ahora vamos a entrar en la esfera de **qualia**, o de las experiencias intrínsecas (de la palabra latina que signi-

[3] Esta situación puede resultar familiar para los que hayan visto la película de 1999, *The Matrix*.

fica aproximadamente «tales cosas»). Suponga que mediante algún accidente del cableado retinal o neuronal la persona *X* experimenta en rojo el color que la persona *Y* percibe como verde, y viceversa. Entonces cuando ambas vean las luces del semáforo actuarán de la misma manera, pero la *experiencia* que tienen será de alguna manera diferente. Ambas pueden reconocer que el nombre de su experiencia es «la luz es roja», pero las experiencias se sienten de diferente manera. No está claro si eso significa que son los mismos estados mentales o que son estados mentales diferentes.

A continuación retomaremos otro experimento del pensamiento que aborda la cuestión de si los objetos físicos diferentes a las neuronas humanas pueden tener estados mentales.

El experimento de la prótesis cerebral

El experimento de la prótesis cerebral fue introducido por Clark Glymour a mediados de los años 70, y retocado posteriormente por John Searle (1980), aunque se asocia más comúnmente al trabajo de Hans Moravec (1988). Trata de lo siguiente: suponga que la neurofisiología ha evolucionado hasta tal punto que el comportamiento y las conexiones de entrada y salida de todas las neuronas del cerebro humano se entienden perfectamente. Además, suponga que podemos construir mecanismos electrónicos microscópicos que imitan este comportamiento y que pueden interconectarse con el tejido neuronal. Y finalmente, suponga que una técnica quirúrgica milagrosa puede sustituir las neuronas individuales con los mecanismos electrónicos sin interrumpir el funcionamiento del cerebro por completo. El experimento consiste en sustituir gradualmente todas las neuronas de la cabeza de alguien con mecanismos electrónicos y a continuación invertir el proceso para retornar al sujeto a su estado biológico normal.

Nos preocupa tanto el comportamiento externo como la experiencia interna del sujeto, durante y después de la operación. Por definición del experimento, el comportamiento externo del sujeto no debe sufrir ningún cambio en comparación con lo que se observaría si la operación no se llevase a cabo[4]. Ahora bien, aunque la presencia o ausencia de consciencia no la pueda asegurar fácilmente un tercero, el sujeto del experimento debería por lo menos poder registrar cualquier cambio en su propia experiencia consciente. Aparentemente existe una confrontación directa de intuiciones de lo que podría llegar a ocurrir. Moravec, un investigador y funcionalista en robótica, está convencido de que su consciencia no se vería afectada. Searle, un filósofo y naturalista biólogo, también está convencido de que la consciencia desaparecería:

> Ante nuestro asombro encontramos que en efecto estamos perdiendo el control del comportamiento externo. Encontramos, por ejemplo, que cuando los doctores comprueban nuestra visión, dicen: «Hemos puesto un objeto rojo delante de usted; por favor, díganos lo que ve». Desearemos responder gritando «No puedo ver nada. Me he quedado absolutamente ciego». Sin embargo, hablaremos y diremos sin ningún control por nuestra parte: «Observo que hay un objeto rojo delante de mí»... Nuestra experiencia consciente se va reduciendo poco a poco a la nada, mientras que se puede observar externamente que nuestro comportamiento es el mismo. (Searle, 1992)

[4] Imaginemos que utilizamos un sujeto idéntico de «control» que sufre una operación placebo para poder comparar los dos comportamientos.

Sin embargo, podemos hacer algo más que argumentar, a partir de la intuición. En primer lugar, hay que destacar que para que el comportamiento externo siga siendo el mismo, mientras que el individuo se va sumiendo gradualmente en la inconsciencia, debe darse el caso de que la propia voluntad del sujeto se elimine instantáneamente y totalmente; de otra manera, la reducción del conocimiento se vería reflejada en el comportamiento externo, con palabras tales como «Ayuda, mi conocimiento se está reduciendo», u otras palabras con ese mismo efecto. Esta eliminación instantánea de la propia voluntad como resultado de la sustitución gradual de neuronas una a una parece ser una afirmación improbable.

En segundo lugar, examinemos qué ocurre si formulamos al sujeto cuestiones relacionadas con su experiencia consciente durante el período en donde no existen neuronas reales. Por las condiciones del experimento, obtendremos respuestas del tipo, «Me encuentro bien. Debo decir que estoy algo sorprendido porque creía en el argumento de Searle». O de lo contrario, podríamos atizar al sujeto con un bastón acabado en punta y observaríamos la respuesta, «¡Ay, cómo duele!». Ahora bien, durante el curso normal de sucesos, el escéptico puede desechar tales salidas de programas de IA tales como simples artimañas. Ciertamente, es bastante fácil utilizar una regla tal como «Si el sensor 12 identifica "Alto" entonces la salida será "Ay"». Sin embargo lo importante es que, debido a que hemos duplicado las propiedades funcionales de un cerebro humano normal, supondremos que el cerebro electrónico no contiene dichas artimañas. Debemos entonces tener una explicación de las manifestaciones de la consciencia producidas por el cerebro electrónico que atañe sólo a las propiedades funcionales de las neuronas. *Y además, esta explicación debe aplicarse también al cerebro real, el cual tiene las mismas propiedades funcionales*. Parece entonces que existen sólo dos conclusiones posibles:

1. Los mecanismos causales de la consciencia que generan estas clases de salidas en los cerebros normales todavía están funcionando en la versión electrónica, que, por tanto, es consciente.
2. Los sucesos mentales conscientes en el cerebro normal no tienen conexión causal con el comportamiento, y quedan fuera del cerebro electrónico, que no es, por tanto, consciente.

EPIFENOMENAL

Aunque no podamos descartar la segunda posibilidad, esto reduce la consciencia a lo que los filósofos llaman rol **epifenomenal**, algo que ocurre, pero que no deja pistas como si lo fuera en el mundo observable. Además, si la consciencia es en efecto epifenomenal, entonces el cerebro debe tener un segundo mecanismo inconsciente responsable del «Ay».

En tercer lugar, examinemos la situación en que la operación se ha invertido y el sujeto tiene un cerebro normal. Una vez más, el comportamiento externo del sujeto debe ser, por definición, como si la operación no hubiera ocurrido. En particular, deberíamos poder preguntar, «¿Cómo estaba durante la operación?, ¿recuerda el bastón acabado en punta?». El sujeto debe tener recuerdos precisos de la naturaleza real de sus experiencias conscientes, incluyendo las qualia, a pesar del hecho de que según Searle no ha habido tales experiencias.

Searle podría responder que no hemos definido el experimento adecuadamente. Por ejemplo, si las neuronas reales quedan suspendidas entre el momento que se extraen y el tiempo en que se reemplazan en el cerebro, por supuesto entonces no «recordarán»

las experiencias durante la operación. Para tratar esta eventualidad, necesitamos asegurarnos de que el estado de las neuronas se actualiza para reflejar el estado interno de las neuronas artificiales que están reemplazando. Si los supuestos aspectos «no funcionales» de las neuronas reales dan como resultado entonces un comportamiento funcionalmente diferente del que se observa con las neuronas artificiales todavía en su lugar, tenemos un *reductio ad absurdum* simple, debido a que eso significaría que las neuronas artificiales no son equivalentes funcionalmente a las neuronas reales. (*Véase* el Ejercicio 26.3 para una posible refutación de este argumento.)

Patricia Churchland (1986) señala que los argumentos funcionalistas que operan a nivel de neurona también pueden funcionar al nivel de cualquier unidad funcional más grande, un grupo de neuronas, un módulo mental, un lóbulo, una hemisfera, o todo el cerebro. Eso significa que si se acepta la noción de que el experimento de prótesis del cerebro muestra que el cerebro de sustitución es consciente, deberíamos también creer que la consciencia se mantiene cuando el cerebro entero se sustituye por un circuito que hace corresponder la entrada con la salida mediante una tabla de búsqueda. Esto es desconcertante para muchas personas (incluyendo al mismo Turing), que tienen la intuición de que las tablas de búsqueda no son conscientes, o por lo menos, que las experiencias concientes generadas durante la búsqueda en tabla no son las mismas que las generadas durante la operación de un sistema que se podría describir (incluso en un sentido computacional ingenuo) como accediendo y generando ideas, introspecciones, metas, etc. Esto podría sugerir que el experimento de la prótesis del cerebro no puede utilizar un reemplazo de todo el cerebro en conjunto, si ha de ser efectivo al guiar las intuiciones, pero esto no significa que se debe utilizar el reemplazo de un átomo cada vez, tal como Searle nos hace creer.

La habitación china

Nuestro experimento final del pensamiento es quizás el más famoso de todos. Este se debe a John Searle (1980), quien describe un sistema hipotético que claramente está ejecutando un programa y que pasa la prueba de Turing, pero que igualmente y de manera clara no *entiende* (según Searle) ninguna entrada ni salida. Su conclusión es que ejecutar el programa adecuado (es decir, tener las salidas adecuadas) no es una condición *suficiente* para ser una mente.

El sistema se compone de un hombre, que solamente entiende inglés, y que está equipado con un libro de reglas escrito en inglés y varias pilas de papel, algunas en blanco y algunas con inscripciones indescifrables. (El hombre entonces hace el papel de la CPU, el libro de normas es el programa y los papeles son el dispositivo de almacenamiento.) El sistema se encuentra dentro de una habitación con una apertura al exterior. A través de la apertura se van deslizando los papeles con símbolos indescifrables. El hombre encuentra los símbolos correspondientes en el libro de reglas y sigue las instrucciones. Las instrucciones pueden incluir escritura de símbolos en los papeles nuevos que van saliendo, encontrar símbolos en las pilas de papeles, reorganizar las pilas, etc. Finalmente las instrucciones harán que un símbolo o más sean transcritos a un trozo de papel que se pasa otra vez al mundo exterior.

Hasta ahora todo está bien. Pero desde fuera, se observa un sistema que está sacando la entrada en forma de sentencias chinas y generando respuestas chinas que obviamente son tan «inteligentes» como las de la conversación imaginada por Turing[5]. Entonces Searle argumenta lo siguiente: la persona que está en la habitación no entiende el chino (supuestamente). El libro de reglas y las pilas de papel, que son sólo trozos de papel no entienden el chino. Por consiguiente no está habiendo comprensión del chino. *De aquí que, según dice Searle, ejecutar el programa adecuado no genera necesariamente entendimiento*.

Al igual que Turing, Searle examinó e intentó rechazar una serie de respuestas a este argumento. Algunos comentaristas, incluyendo a John McCarthy y a Robert Wilensky, propusieron lo que Searle llama la respuesta de los sistemas. La objeción es que aunque se pueda preguntar si el hombre de la habitación entiende el chino, esto es como preguntar si la CPU puede admitir raíces cúbicas. En ambos casos, la respuesta es no, y en ambos casos según la respuesta del sistema, el sistema completo *sí* tiene la capacidad en cuestión. Ciertamente, si se pregunta a la habitación china si entiende el chino, la respuesta sería afirmativa (en un chino fluido). Según la convención educada de Turing, esto debería ser suficiente. La respuesta de Searle es reiterar la cuestión de que el entendimiento no está en el hombre y no puede estar en el papel, de forma que no puede haber entendimiento. También sugiere algo más: que nos podríamos imaginar al hombre memorizando el libro de reglas y el contenido de todas las pilas de papel, de manera que no hubiera nada que tener que entender *excepto* al hombre; y una vez más, si se pregunta al hombre (en inglés), la respuesta será negativa.

En este punto hemos topado con los temas reales. El cambio a la memorización desde el papel es una pista falsa, porque ambas formas son simplemente instanciaciones físicas de un programa en ejecución. La afirmación real que hace Searle se basa en los cuatro axiomas siguientes (Searle, 1990):

1. Los programas informáticos son entidades sintácticas y formales.
2. Las mentes tienen contenidos o semánticas mentales.
3. La sintaxis por sí misma no es suficiente para la semántica.
4. Los cerebros originan las mentes.

A partir de los tres primeros axiomas, Searle concluye diciendo que los programas no son suficientes para las mentes. En otras palabras, un agente que ejecuta un programa podría ser una mente, pero no es necesariamente una mente sólo por virtud de ejecutar programas. A partir de los cuatro axiomas concluye que «Cualquier otro sistema capaz de originar mentes tendría que tener poderes causales (por lo menos) equivalentes a los de los cerebros». De aquí infiere que cualquier cerebro artificial tendría que duplicar los poderes causales, no sólo ejecutar un programa en particular, y que los cerebros humanos no producen fenómenos mentales sólo por virtud de ejecutar un programa.

[5] El hecho de que las pilas de papel podrían ser mucho más grandes que todo el planeta y de que la generación de respuestas llevaría millones de años no redunda en la estructura *lógica* del argumento. Un propósito de la formación filosófica es desarrollar un sentido bien afinado de las objeciones que guardan relación y las que no.

Las conclusiones de que los programas no son suficientes para las mentes *sí* que proceden de estos axiomas, si somos generosos en su interpretación. Pero la conclusión no es satisfactoria: todo lo que Searle ha demostrado es que si explícitamente negamos el funcionalismo (eso es lo que hace su axioma (3)) entonces no podemos concluir necesariamente diciendo que los no-cerebros son mentes. Esto es bastante razonable, de manera que el argumento completo se reduce si el axioma (3) puede ser aceptado. Según Searle, el punto de vista del argumento de la habitación china es proporcionar intuiciones al axioma (3). Pero la reacción de este argumento muestra que proporciona intuiciones sólo a los que ya se han inclinado a aceptar la idea de que meros programas no pueden generar un entendimiento de verdad.

Reiterando, el objetivo del argumento de la habitación china es refutar la IA fuerte, es decir la afirmación de que ejecutar la clase adecuada de programa da como resultado necesariamente una mente. Esto se hace exhibiendo un sistema aparentemente inteligente que ejecuta la clase adecuada de programa que no es, según Searle, *demostrablemente* una mente. En este sentido, Searle apela a la intuición, no a la prueba: sólo hay que observar la habitación; ¿qué hay en ella que sea una mente? Sin embargo, se podría hacer el mismo argumento sobre el cerebro: observe el conjunto de células (o de átomos), que funcionan ciegamente según las leyes de la Bioquímica (o la Física): ¿Qué hay ahí que sea una mente? ¿Por qué un trozo de cerebro puede ser una mente mientras que un trozo de hígado no puede serlo?

Además, cuando Searle admite que materiales diferentes a las neuronas podrían en principio ser una mente, debilita su argumento aún más por dos razones: en primer lugar sólo tenemos las intuiciones de Searle (o las nuestras propias) para decir que la habitación china no es una mente, y en segundo lugar, aunque determinemos que la habitación no es una mente, no nos dice nada sobre si un programa que se ejecuta en algún otro medio físico (incluyendo un computador) podría ser una mente.

Searle permite la posibilidad lógica de que el cerebro esté implementando de verdad un programa de IA de la clase tradicional. Sin embargo, si el mismo programa se ejecuta en la clase inadecuada de máquina no sería una mente. Searle no cree que las «máquinas no puedan tener mentes», en cambio afirma que algunas máquinas *sí* que tienen mentes, los hombres son máquinas biológicas con mentes. No nos queda ninguna guía para ver qué tipos de máquinas califican o no califican.

26.3 La ética y los riesgos de desarrollar la Inteligencia Artificial

Hasta ahora nos hemos concentrado en si *podemos* desarrollar la IA, pero debemos también tener en cuenta si *deberíamos* hacerlo. Si es más probable que los efectos de la tecnología de la IA sean más negativos que positivos, sería responsabilidad moral de los trabajadores en su campo redirigir su investigación. Muchas de las nuevas tecnologías han tenido efectos negativos no intencionados: el motor de combustión trajo la polución ambiental y la pavimentación del paraíso; la fisión nuclear produjo el desastre de

Chernobyl, la Isla de las Tres Millas y la amenaza de la destrucción mundial. Todos los científicos e ingenieros se enfrentan a consideraciones éticas de cómo deberían actuar en el trabajo, qué proyectos deberían o no deberían hacer, cómo los deberían abordar. Existe incluso un manual sobre la Ética de los Computadores (*Ethics of Computing,* Berleur y Brunnstein, 2001). Sin embargo, la IA parece que expone problemas nuevos yendo más allá de, por ejemplo, construir puentes que no se desmoronen:

- Las personas podrían perder sus trabajos por la automatización.
- Las personas podrían tener demasiado (o muy poco) tiempo de ocio.
- Las personas podrían perder el sentido de ser únicos.
- Las personas podrían perder algunos de sus derechos privados.
- La utilización de los sistemas de IA podría llevar a la pérdida de responsabilidad.
- El éxito de la IA podría significar el fin de la raza humana.

Vamos a examinar en orden cada uno de estos temas.

Las personas podrían perder sus trabajos por la automatización. La economía industrial moderna ha llegado a depender en general de los computadores, y selecciona programas de IA en particular. Por ejemplo, gran parte de la economía, especialmente en Estados Unidos, depende de la disponibilidad de créditos del consumidor. Las aplicaciones de las tarjetas de crédito, aceptaciones de cargos y detección de fraudes se realizan mediante programas de IA. Se podría decir que miles de trabajadores han sido desplazados por estos programas de IA, pero en efecto si eliminamos estos programas de IA estos trabajos no existirían, porque la mano de obra humana añadiría un coste adicional a las transacciones. Hasta ahora, la automatización por medio de la tecnología de la IA ha creado más trabajos de los que ha eliminado, y ha creado puestos de trabajo más interesantes y mejor pagados. Ahora que el programa IA canónico es un «agente inteligente» diseñado para ayudar a un hombre, la pérdida de trabajo preocupa menos que cuando la IA se centraba en los sistemas expertos diseñados para sustituir a los hombres.

Las personas podrían tener demasiado (o muy poco) tiempo de ocio. Alvin Toffler en su libro *Future Shock* (1970) escribió, «La semana laboral se ha recortado el 50 por ciento a finales del siglo. No es absurdo decir que se volverá a reducir a la mitad hacia el año 2000». Arthur C. Clarke (1968b) escribió que las personas del año 2001 «podrían enfrentarse a un futuro de gran aburrimiento, en donde el problema principal de la vida es decidir cual de los cientos de canales de la televisión seleccionar». La única predicción que ha llegado a resultar cierta es el número de canales de televisión (Springsteen, 1992). En cambio, las personas que trabajan en las industrias muy relacionadas con el conocimiento han descubierto que forman parte de un sistema computerizado integrado que funciona 24 horas al día; para mantenerlo se han visto forzados a trabajar durante *más* horas. En una economía industrial, las recompensas son aproximadamente proporcionales al tiempo invertido; trabajar el 10 por ciento más llevaría a producir un incremento del 10 por ciento en los ingresos. En una economía de la información marcada por la comunicación de un ancho de banda alto y por una reproducción fácil de la propiedad intelectual (lo que Frank y Cook (1996) llaman «Winner-Take-All Society» (la Sociedad del ganador se hace con todo)), existe una gran recompensa por ser ligeramente mejor que la competencia; trabajar un 10 por ciento más podría significar un 100 por 100 de incremento en los ingresos. De forma que todos se sienten más presionados por trabajar más fuerte. La IA

incrementa el ritmo de la innovación tecnológica y contribuye así a esta tendencia general, pero la IA también mantiene la promesa de permitirnos ahorrar tiempo y permitir que nuestros agentes automatizados hagan las cosas por un tiempo.

Las personas podrían perder su sentido de ser únicas. En el libro de Weizenbaum (1976), *Computer Power and Human Reason* (El poder de los computadores y la razón humana), el autor del programa ELIZA, señala algunas de las posibles amenazas que supone la IA para la sociedad. Uno de los argumentos principales de Weizenbaum es que la investigación en IA hace posible la idea de que los hombres sean autómatas, una idea que produce pérdida de autonomía o incluso de humanidad. Podemos señalar que la idea ya existía mucho antes que la IA, nos podemos remontar a *L'Homme Machine* (La Mettrie, 1748). También destacaremos que la humanidad ha sobrevivido a otros contratiempos hacia nuestro sentido de la unicidad: *De Revolutionibus Orbium Coelestium* (Copernicus, 1543) desplazó a la tierra lejos del centro del sistema solar, y *Descent of Man* (Darwin, 1871) puso al *homo sapiens* al mismo nivel que otras especies. La IA, aunque sea una materia de gran éxito, quizá sea por lo menos amenazante para las suposiciones morales de la sociedad del siglo XXI al igual que la teoría de la evolución lo fue para los del siglo XIX.

Las personas podrían perder algo de sus derechos privados. Weizenbaum también señaló que la tecnología del reconocimiento de voz podría llevar a una intercepción extensa de cableados, y de aquí a la pérdida de las libertades civiles. No previó un mundo con amenazas terroristas que cambiarían el equilibrio de la vigilancia que estarían dispuestas a aceptar las personas, sino que reconocería correctamente que la IA tiene el potencial de producir una vigilancia en grandes cantidades. Esta predicción puede convertirse en realidad: el sistema clasificado Echelon del gobierno americano «consiste en una red de envíos de escucha, campos de antenas y estaciones de radar; el sistema está respaldado por computadores que utilizan traducción de lenguajes, reconocimiento de voz y palabras clave que buscan pasar por la criba automáticamente todo el tráfico de llamadas telefónicas, correos electrónicos, faxes y telex»[6]. Algunas personas reconocen que la computerización conduce a la pérdida de privacidad, por ejemplo, el consejero delegado de Sun Microsystems, Scott McNealy ha dicho que «De cualquier forma tenemos privacidad cero. Hay que superarlo». Otros no están de acuerdo. Por ejemplo, el juez Louis Brandeis en 1890 escribió que «La privacidad es el derecho más completo y extenso de todos... el derecho de la personalidad de uno mismo».

La utilización de sistemas de IA podría llevar a la pérdida de responsabilidad. En la atmósfera de litigios que prevalece en Estados Unidos, la obligación legal se convierte en un tema importante. Cuando un médico depende del juicio de un sistema médico experto para hacer diagnóstico, ¿quién es el culpable si el diagnóstico es erróneo? Afortunadamente, debido en parte a la mayor influencia en medicina de métodos teóricos para las decisiones, se acepta que la negligencia no puede demostrarse si un médico lleva cabo procedimientos médicos que tienen una utilidad altamente *esperada*, incluso si el resultado *real* es catastrófico para el paciente. Por tanto la pregunta debería ser «¿Quién tiene la culpa si el diagnóstico no es razonable?». Hasta ahora, los juzgados han man-

[6] *Véase* «Eavesdropping in Europe» (Escucha a escondidas en Europa), *Wired* news, 30/09/1998, e informes citados por la UE.

tenido que los sistemas médicos expertos desempeñan el mismo papel que los libros de texto médicos y los libros de referencias; los médicos son responsables de entender el razonamiento que soporta esta decisión y de utilizar su propio juicio a la ahora de decidir si se aceptan las recomendaciones del sistema. Por tanto, en el diseño de sistemas médicos expertos como agentes, no se debería considerar que las acciones afectan directamente a los pacientes, sino que influyen directamente en el comportamiento del médico. Si los sistemas expertos se hacen más fiables y precisos que los hombres que hacen los diagnósticos, los médicos podrían tener obligaciones legales si *no* utilizan las recomendaciones de un sistema experto. Gawande (2002) explora esta premisa.

Están empezando a aparecer temas similares en relación con la utilización de agentes inteligentes en Internet. Se han hecho progresos en la incorporación de limitaciones en los agentes inteligentes de forma que no pueden, por ejemplo, dañar los archivos de otros usuarios (Weld y Etzioni, 1994). El problema se magnifica cuando el dinero cambia de manos. Si las transacciones monetarias las realiza un agente inteligente en nombre de alguien, ¿está obligado por las deudas incurridas? ¿Sería posible que un agente inteligente tuviera activos o que realizara compras electrónicas en su propio nombre? Hasta ahora, parece que estas cuestiones no se entienden de forma clara. En nuestro conocimiento, ningún programa ha recibido ningún estado legal como individuo con fines financieros; por el momento, parece que no es razonable hacerlo. Los programas tampoco son considerados «conductores» para reforzar las regulaciones del tráfico en autovías reales. En las leyes californianas, por lo menos, no parece haber ninguna sanción legal que impida a un vehículo automatizado exceder los límites de velocidad, aunque el diseñador del mecanismo de control de vehículos sí tuviera obligación en caso de accidente. Al igual que con la tecnología para la reproducción humana, la ley tiene todavía que ponerse a la altura de los nuevos desarrollos.

El éxito de la IA podría significar el fin de la raza humana. Casi cualquier tecnología tiene el potencial de hacer daño si se encuentra en las manos equivocadas, pero con la IA y la robótica, tenemos el problema nuevo de que las manos equivocadas podrían pertenecer a dicha tecnología. Incontables historias de ciencia ficción nos han alertado de los robots o de los «cyborgs» (robots-hombre) que se comportan de forma enajenada. Entre los primeros ejemplos se incluyen *Frankenstein* de Mary Shelly, o el *Modern Prometheus* (*El Prometeo Moderno)* (1818)[7], y la obra de Karel Capek *R.U.R.* (1921), en la que los robots conquistan el mundo. En cine, también tenemos Terminador (*The Terminator*) (1984), en donde se combinan clichés de robots que conquistan el mundo con el viajar en el tiempo, y Matrix (*The Matrix*) (1999), en donde se combinan robots que conquistan el mundo y el cerebro en una cubeta.

Para una gran mayoría, parece que los robots son protagonistas de muchas historias de conquistas del mundo porque representan lo desconocido, de la misma manera que las brujas y los fantasmas de cuentos de otras décadas más antiguas. ¿Suponen una amenaza más creíble que las brujas y los fantasmas? Si los robots estuvieran diseñados adecuadamente como agentes que adoptan las metas de sus propietarios, probablemente no supondrían una amenaza: los robots que derivan de mayores avances sobre sus diseños actuales, sí que van a servir y no a conquistar. Los hombres utilizan su inteligencia de

[7] Charles Babbage, cuando era joven, se vio influenciado por la lectura de *Frankenstein*.

formas agresivas porque son innatas, por naturaleza. Sin embargo, las máquinas que construimos no tienen que ser innatamente agresivas, a menos que decidamos que así sean. Por un lado, es posible que los computadores logren una clase de conquista sirviendo y haciéndose indispensables, de la misma manera que los automóviles han conquistado en cierto sentido el mundo industrializado. Este escenario merece más atención. En 1965, I. J. Good escribió:

> Vamos a definir una máquina ultrainteligente como una máquina que puede sobrepasar con mucho todas las actividades intelectuales de cualquier hombre, por muy inteligente que sea. Puesto que el diseño de las máquinas es una de estas actividades intelectuales, una máquina ultrainteligente podría diseñar máquinas incluso mejores; entonces existiría incuestionablemente una «explosión de inteligencia», y la inteligencia del hombre quedaría bastante atrás. Así pues la primera máquina ultrainteligente es la última invención que haya hecho nunca el hombre, siempre que la máquina sea lo suficientemente dócil como para decirnos cómo controlarla.

SINGULARIDAD TECNOLÓGICA

La «explosión de inteligencia» también ha sido llamada **singularidad tecnológica** por el profesor de Matemáticas y autor de ciencia ficción Vernor Vinge, (1993) quien escribió que, «Dentro de treinta años, tendremos el medio tecnológico de crear una inteligencia superhumana. Algún tiempo después, la era humana habrá acabado». Good y Vinge (y muchos otros) señalan correctamente que la curva del progreso tecnológico actualmente está creciendo de manera exponencial (tomemos en consideración, por ejemplo, la ley de Moore). Sin embargo, es un buen paso adelante extrapolar que la curva continuará hacia la singularidad de un crecimiento casi infinito. Hasta ahora, el resto de tecnologías han seguido una curva en forma de S, en donde el crecimiento exponencial finalmente disminuye.

La preocupación y el miedo de Vinge radica en llegar a la singularidad, sin embargo otros científicos y futuristas gozan con esa idea. En el *Robot, Mere Machine to Trascendent Mind* (Mera máquina hacia la mente trascendente), Hans Moravec predice que los robots se igualarán a la inteligencia humana en 50 años y a continuación la excederán:

> De manera bastante rápida podríamos quedar desplazados y fuera de la existencia. No estoy tan alarmado como muchos otros por esta última posibilidad, ya que considero que las máquinas del futuro son nuestra progenie, «hijos de mente» construidos a nuestra imagen y semejanza, es decir, nosotros mismos pero en una forma más potente. Al igual que los hijos biológicos de generaciones anteriores, representarán la mejor esperanza de la humanidad para un futuro a largo plazo. Nos corresponde a nosotros ofrecerles todas las ventajas, y cómo retirarnos cuando ya no podamos contribuir. (Moravec, 2000.)

Ray Kurzweil, en *The Age of Spiritual Machines* (2000) predice que hacia el año 2099 existirá «una fuerte tendencia hacia una fusión del pensamiento humano en el mundo de la inteligencia de la máquina que las especies humanas crearon inicialmente. Ya no existe una distinción clara entre los hombres y los computadores». Existe incluso una palabra nueva, **trashumanismo**, que se refiere al movimiento social real que ansía este futuro. Basta con decir que estos temas presentan un reto para la mayoría de los teóricos que consideran la preservación de la vida humana y de las especies como algo bueno.

TRASHUMANISMO

Finalmente, tomemos en consideración el punto de vista del robot. Si los robots adquieren consciencia, tratarlos entonces como meras «máquinas» (por ejemplo, tratarlos como algo aparte) podría ser inmoral. Los robots también deben actuar moralmente,

necesitaríamos programarlos con una teoría de lo que está bien y lo que está mal. Los escritores de ciencia ficción han abordado el tema de los derechos y responsabilidades de los robots, empezando por Isaac Asimov (1942). La famosa película titulada *A.I.* (Spielberg, 2001) se basó en una historia de Brian Aldiss sobre el robot inteligente que fue programado para creer que él era un humano y que no entiende el abandono final por parte de su madre-propietaria. La historia (y la película) convencen de la necesidad de los robots para moverse por sus derechos civiles.

26.4 Resumen

Este capítulo ha abordado los siguientes temas:

- Los filósofos utilizan el término **IA débil** para la hipótesis de que las máquinas podrían comportarse posiblemente de forma inteligente, y el término **IA fuerte** para la hipótesis de que dichas máquinas contarían como si tuvieran mentes reales (en contraposición a las mentes simuladas).
- Alan Turing rechazó la cuestión «¿Pueden pensar las máquinas?», y lo sustituyó por un test de comportamiento. Previeron muchas objeciones a la posibilidad de máquinas pensantes. Pocos investigadores en IA prestan atención al Test de Turing, y prefieren concentrarse en el rendimiento de sus sistemas en base a tareas prácticas, en vez de la habilidad de imitar a los humanos.
- Actualmente existe el acuerdo general de que los estados mentales son los estados del cerebro.
- Los argumentos a favor y en contra de la IA fuerte no son concluyentes. Unos pocos investigadores dominantes en la disciplina de la IA creen que cualquier cosa significativa gira en torno al resultado del debate.
- La consciencia sigue siendo un misterio.
- Se han identificado ocho amenazas potenciales para la sociedad que se exponen tanto ante la IA como ante una tecnología relacionada. Podemos concluir diciendo que algunas amenazas son improbables, pero merece la pena revisar dos de ellas en particular. La primera es que las máquinas ultrainteligentes podrían llevarnos a un futuro muy diferente del actual y puede que no sea de nuestro agrado. La segunda es que la tecnología de la robótica puede permitir a individuos con una psicopatía emplear armas de destrucción masiva. Concluimos diciendo que esto es más una amenaza de la biotecnología y nanotecnología que de la robótica.

Notas bibliográficas e históricas

La naturaleza de la mente ha sido un tema estándar de la teorización filosófica desde tiempos inmemoriales hasta hoy en día. En el *Phaedo*, Platón consideró y rechazó de forma específica la idea de que la mente podría ser una «adaptación» o un modelo de organización de las partes del cuerpo, un punto de vista que se aproxima al punto de vista fun-

cionalista de la filosofía moderna de la mente. En cambio, determinó que la mente tenía que ser un alma inmortal, inmaterial, separable del cuerpo y diferente en sustancia, del punto de vista del dualismo. Aristóteles distinguió una variedad de almas (en griego $\psi\upsilon\chi\eta$) de los seres vivos, describiendo algunas, por lo menos, de forma funcionalista. (*Véase* Nussbaum (1978) para más información sobre el funcionalismo de Aristóteles.)

Descartes destaca por su punto de vista dualista de la mente humana, aunque irónicamente su influencia histórica se dirigió hacia el mecanismo y el materialismo. Explícitamente, concibió a los animales como autómatas, y se anticipó al Test de Turing, escribiendo que «no se puede concebir [que una máquina] elabore diferentes colocaciones de palabras para dar una respuesta con un significado adecuado para todo lo que se diga en su presencia, incluso de la misma forma que los hombres más aburridos cuando hablan» (Descartes, 1637). La ardiente defensa que hace Descartes del punto de vista de los animales como autómatas realmente tuvo el efecto de facilitar también la concepción de los hombres como autómatas, incluso aunque él mismo no diera ese paso. El libro *L'Homme Machine* o *Man a Machine* (La Mettrie, 1748) argumentó explícitamente que los hombres son autómatas.

La filosofía analítica moderna normalmente ha aceptado el materialismo (a menudo mediante la **teoría de la identidad** del estado del cerebro (Place, 1956; Armstrong, 1968), la cual afirma que los estados mentales son idénticos a los estados del cerebro)), sin embargo ha estado mucho más dividida en el funcionalismo, en la analogía de la máquina con le mente humana, y la cuestión de si las máquinas literalmente pueden pensar. Scriven (1953), por ejemplo, entre las primeras respuestas filosóficas a Turing (1950) «Computing Machinery and Intelligence», intentó negar que tuviera significado decir que las máquinas pudieran pensar, basándose en que dicha afirmación violaba el significado del mundo. En el apéndice de la reimpresión de este artículo se puede ver que, por lo menos, Scriven hacia el año 1963 se retractó de este punto de vista (Anderson, 1964). El científico informático Edsger Dijkstra dije que «La cuestión de si un computador puede pensar no es más interesante que la cuestión de si un submarino puede nadar». Ford y Hayes (1995) argumentan que el Test de Turing no es útil para la IA.

El funcionalismo es la filosofía de la mente que más naturalmente sugiere la IA, y las críticas del funcionalismo a menudo toman forma a partir de las críticas sobre la IA (como en el caso de Searle). Siguiendo la clasificación utilizada por Block (1980), se pueden distinguir variedades del funcionalismo. La **teoría de la especificación funcional** (Lewis, 1966, 1980) es una variante de la teoría de la identidad del estado del cerebro la cual selecciona los estados del cerebro que se identifican con los estados mentales en su papel (rol) funcional. La **teoría de la identidad del estado funcional** (Putnam, 1960, 1967) se basa más estrechamente en la analogía de la máquina. No identifica los estados mentales con los estados *físicos* del cerebro sino con los estados abstractos computacionales del cerebro concebidos expresamente como un dispositivo informático. Se supone que estos estados abstractos son independientes de la composición física específica del cerebro, lo que lleva a considerar que la teoría de la identidad del estado funcional es una forma de dualismo.

Tanto la teoría de la identidad del estado del cerebro como las diferentes formas de funcionalismo han sido atacadas por autores afirmando que no tienen en consideración la *qualia* o el aspecto «a qué se asemejan» de los estados mentales (Nagel, 1974). En

cambio, Searle se ha centrado en la supuesta incapacidad del funcionalismo (Searle, 1980, 1984, 1992). Churchland y Churchland (1982) rebaten ambas formas de crítica.

MATERIALISMO ELIMINATIVO

El **materialismo eliminativo** (Rorty, 1965; Churchland, 1979) difiere de las otras teorías prominentes de la filosofía de la mente, en que no intenta dar cuenta de por qué son verdaderas las ideas de «la psicología popular» y del sentido común sobre la mente, sino que por el contrario las rechaza como si fueran falsas e intenta sustituirlas por una teoría puramente científica de la mente. En principio, esta teoría científica podría ser proporcionada por una IA clásica, pero en cambio, en la práctica, los materialistas eliminativos tienden a apoyarse en la investigación de la neuro ciencia y de las redes neuronales (Churchland, 1986), sin embargo, basándose en la IA clásica, especialmente en la investigación de la «representación del conocimiento» del tipo descrito en el Capítulo 10, tienden a confiar en la verdad de la psicología popular. Aunque el punto de vista del «estado intencional» (Dennett, 1971) pudiera interpretarse como funcionalista, probablemente debería considerarse como una forma de materialismo eliminativo, en el sentido de que la «postura intencional» no tiene que reflejar ninguna propiedad objetiva del agente hacia el que se toma esa postura. También se debería destacar que es posible ser un materialista eliminativo en algunos aspectos de la mentalidad a la vez que se analizan otras de forma diferente. Por ejemplo, Dennett (1978) es mucho más eliminativo en cuanto a la *qualia* que en cuanto a la intencionalidad.

Ya se han proporcionado en este mismo capítulo otras fuentes de críticas importantes hacia la IA débil. Aunque estuvo muy en auge ridiculizar los enfoques simbólicos en la era de las redes post-neuronales, no todos los filósofos son críticos de GOFAI. De hecho, algunos son ardientes partidarios e incluso buenos profesionales. Zenon Pylyshyn (1984) ha argumentado que la cognición se puede entender mejor a través de un modelo computacional, no sólo en principio sino también en la forma de realizar una investigación contemporánea, y ha refutado específicamente las críticas de Dreyfus en cuanto al modelo computacional de la cognición humana (Pylyshyn, 1974). Gilbert Harman (1983), quien en el análisis de la revisión de las creencias ha realizado conexiones con la investigación de la IA en los sistemas del mantenimiento de verdad. Michael Bratman ha aplicado su modelo de «creencia-deseo-intención» de la psicología humana (Bratman, 1987) a la investigación de la IA en planificación (Bratman, 1992). En el lado extremo de la IA fuerte, Aaron Sloman (1978, p. xiii) ha descrito incluso como «racista» el punto de vista de Joseph Weizenbaum (Weizenbaum, 1976) diciendo que las máquinas inteligentes hipotéticas no deberían considerarse personas.

La literatura filosófica sobre las mentes, los cerebros, y otros temas relacionados es extensa y algunas veces difícil de leer sin una práctica adecuada en la terminología y en los métodos de argumentar empleados. La Enciclopedia de la Filosofía (*Encyclopedia of Philosofy*) (Edwards, 1967) es una ayuda sorprendentemente fidedigna y útil de este proceso. El Diccionario de Filosofía de Cambridge (*The Cambridge Dictionary of Philosofy*) (Audi, 1999) es un trabajo más reducido y más accesible, pero las entradas principales (tales como «filosofía de la mente») todavía abarcan 10 páginas o más. La Enciclopedia de la Ciencia Cognitiva del MIT (*MIT Encyplopedia of Cognitive Sciencie*) (Wilson y Keil, 1999) aborda la filosofía de la mente así como la biología y la psicología de la mente. Otras colecciones de artículos sobre filosofía de la mente, incluyendo el funcionalismo y otros puntos de vista relacionados con la IA son El Materialismo y el Problema de Men-

te-Cuerpo (*Materialism and the Mind-Body Proble,* Rosenthal, 1971) y Lecturas de la Filosofía de la Psicología (*Readings in the Philosophy of Psycology*, volumen 1, Block, 1980). Biro y Shahan (1982) presentan una colección dedicada a los pros y los contras del funcionalismo. Entre las antologías que tratan más específicamente con la relación entre la Filosofía y la IA se incluyen *Minds and Machines* (Anderson, 1964), *Philosophical Perspectives in Artificial Intelligenc*e (Ringle, 1979), *Mind Design* (Haugeland, 1981) y *The Philosophy of Artificial Intelligence* (Boden, 1990). Existen varias introducciones a la «cuestión IA» filosófica (Boden, 1977, 1990; Haugeland, 1985; Copeland, 1993). *The Behavioral and Brain Sciences*, abreviado *BBS*, es una publicación importante dedicada a debates filosóficos y científicos sobre la IA y la neurociencia. En publicaciones como *AI and Society*, *Law, Computers and Artificial Intelligence*, y *Artificial Intelligence and Law* se tratan los temas de la ética y la responsabilidad en la IA.

EJERCICIOS

26.1 Repase la lista de presuntas incapacidades de las máquinas de Turing, identificando cuáles se han logrado, cuáles se pueden lograr en principio por cualquier programa, y cuáles son aún problemáticas porque requieren estados mentales conscientes.

26.2 ¿La refutación de la habitación china demuestra necesariamente que los computadores con una programación adecuada tienen estados mentales? ¿La aceptación del argumento significa necesariamente que los computadores no pueden tener estados mentales?

26.3 En el argumento de la prótesis del cerebro, es importante poder restaurar el cerebro de un sujeto a un estado normal, de tal manera que su comportamiento externo sea como si la operación no hubiera tenido lugar. ¿Puede un escéptico objetar razonablemente que esto requeriría actualizar las propiedades neuropsicológicas de las neuronas que se relacionan con la experiencia consciente, distintas de las que tienen que ver con el comportamiento funcional de las neuronas?

26.4 Encuentre y analice un caso relacionado con los medios de comunicación populares sobre un argumento o más, en relación con la idea de que la IA es imposible.

26.5 Intente definir los términos «inteligencia», «pensamiento» y «consciencia». Sugiera algunas posibles objeciones a sus definiciones.

26.6 Analice las posibles amenazas a la sociedad de la tecnología de IA. ¿Cuáles son las amenazas más serias, y cómo se podrían combatir? ¿Cómo se podrían comparar con las posibles ventajas?

26.7 ¿Cómo se pueden comparar las posibles amenazas de la tecnología de la IA con otras de otras tecnologías informáticas, y con las de las tecnologías nucleares, bionucleares y nanonucleares?

26.8 Algunos críticos piensan que la IA es imposible, mientras que otros dicen que es *demasiado* posible, y que las máquinas ultrainteligentes suponen una amenaza. ¿Cuál de estas objeciones piensa que es más probable? ¿Sería una contradicción mantener ambas opiniones?

27 IA: presente y futuro

En en el que nos interesamos por dónde nos encontramos y hacia dónde vamos, lo cual es bueno antes de seguir hacia adelante.

En la Parte I, propusimos una visión unificada de la IA como diseño racional de agentes. Demostramos que el problema del diseño depende de las percepciones y acciones de los que disponga ese agente, las metas que debería satisfacer el comportamiento del agente, y la naturaleza del entorno. Pueden existir diferentes diseños de agentes, desde agentes reactivos hasta agentes basados en conocimiento y completamente deliberativos. Además, los componentes de estos diseños pueden tener diferentes instanciaciones, por ejemplo, lógicas, probabilísticas o «neuronales». En los capítulos intermedios se presentaron los principios por los que funcionan estos componentes.

Ha habido un tremendo avance tanto en el entendimiento científico como en nuestras habilidades tecnológicas en lo referente a los diseños y componentes de agentes. En este capítulo, no entraremos en detalles y nos preguntaremos, *¿Nos va a llevar este progreso a un agente inteligente de propósito general que pueda actuar correctamente en diferentes entornos?* El Apartado 27.1 estudia los componentes de un agente inteligente para evaluar lo que se conoce y lo que no. El Apartado 27.2 hace lo mismo, pero en la arquitectura completa de los agentes. El Apartado 27.3 pregunta si «el diseño de agentes racionales» es el objetivo adecuado en primer lugar. (La respuesta es, «Realmente no, pero por ahora está bien».) Finalmente, el Apartado 27.4 examina las consecuencias del éxito en nuestros esfuerzos.

27.1 Componentes de los agentes

En el Capítulo 2 presentamos varios diseños de agentes y sus componentes. Para centrar nuestro estudio, examinaremos el agente basado en la utilidad, como se muestra en la Figura 27.1. Este es el diseño de agentes más general de todos. Estudiaremos también su extensión con capacidades de aprendizaje, como se representa en la Figura 2.15.

Figure 27.1 Un agente basado en utilidades y en modelos, tal como se presentó primero en la Figura 2.14.

Interacción con el entorno a través de sensores y actuadores: durante mucho tiempo en la historia de la IA, esto ha sido un notorio punto débil. Con unas pocas excepciones honorables, los sistemas IA se construyeron de tal forma que los humanos tenían que proporcionar las entradas e interpretar las salidas, mientras que los sistemas robóticos se centraban en tareas de bajo nivel en donde el razonamiento y la planificación de alto nivel estaban totalmente ausentes. Esto fue debido en parte al gran esfuerzo que requería la ingeniería y el alto coste para hacer funcionar robots listos. En los últimos años la situación ha cambiado rápidamente con la disponibilidad de robots programables listos para utilizarse o prefabricados, tales como los robots de cuatro patas que se muestran en la Figura 25.4(b). A su vez, estos se han beneficiado de las pequeñas cámaras CCD baratas y de alta resolución, y de las unidades motoras compactas y fiables. La tecnología MEMS (sistemas micro-electromecánicos) ha facilitado acelerómetros miniaturizados y está elaborando actuadores (efectores) que, por ejemplo, podrán activar a un insecto volador artificial. (También es posible combinar millones de actuadores MEMS para producir actuadores macroscópicos muy potentes.) Para los entornos físicos, entonces, los sistemas IA ya no tienen realmente excusa. Además, ya se dispone de un entorno enteramente nuevo como es Internet.

Seguir la pista del estado del mundo: ésta es una de las capacidades centrales que se requieren para un agente inteligente. Requiere tanto percepción como actualización de las representaciones internas. En el Capítulo 7 describimos los métodos para seguir la pista de los mundos descritos por la lógica proposicional; el Capítulo 10 lo amplió hasta llegar a la lógica de primer orden; y el Capítulo 15 describió los algoritmos de **filtrado** para rastrear entornos inciertos. Las herramientas de filtrado se necesitan cuando está involucrada la percepción real (y por tanto imperfecta). Los algoritmos actuales de filtrado y de percepción pueden combinarse para hacer un trabajo razonable de infor-

mar predicados de bajo nivel, tales como «la taza está sobre la mesa», sin embargo, tenemos mucho que recorrer antes de que puedan informar de que «el Doctor Russell está tomando una taza de té con el Doctor Norvig». Otro problema es que aunque los algoritmos de filtrado aproximados puedan manejar una cantidad bastante grande de entornos, todavía son esencialmente proposicionales; como en la lógica proposicional, no pueden representar objetos ni relaciones de forma explícita. El Capítulo 14, explicó cómo se pueden combinar la probabilidad y la lógica de primer orden para resolver este problema; esperemos que la aplicación de estas ideas para rastrear entornos complejos produzca grandes beneficios. Casualmente, tan pronto como empecemos a hablar de *objetos* y de un entorno incierto, nos encontraremos con la **incertidumbre de identidad**, no conocemos qué objeto es cuál. Este problema ha sido el gran ignorado por la IA basada en la lógica, en la cual se ha supuesto generalmente que las percepciones incorporan símbolos constantes que identifican a los objetos.

INCERTIDUMBRE DE IDENTIDAD

Proyección, evaluación y selección de cursos futuros de acción: los requisitos básicos de representación del conocimiento son los mismos aquí que para seguir la pista del mundo; la dificultad básica es hacer frente a los cursos de acción, tales como tener una conversación o tomar una taza de té, que finalmente constan de miles y millones de pasos primitivos para un agente real. Es sólo al imponer una **estructura jerárquica** que los humanos podemos abordarlos. Algunos de los algoritmos de planificación del Capítulo 12 utilizan representaciones jerárquicas y representaciones de primer orden para manejar problemas de esta escala; por otro lado, los algoritmos ofrecidos en el Capítulo 17 para la toma de decisiones bajo incertidumbre están utilizando esencialmente las mismas ideas que los algoritmos de búsqueda basados en el estado del Capítulo 3. Obviamente, aquí todavía queda mucho por hacer, quizás en la línea de los desarrollos actuales en el **aprendizaje de refuerzo jerárquico**.

La utilidad como expresión de preferencias: en principio, basar las decisiones en la maximización de la utilidad esperada es completamente general y evita muchos de los problemas de los enfoques basados puramente en objetivos, tales como objetivos conflictivos y consecución incierta. Sin embargo, hasta ahora, no se ha trabajado mucho en la construcción de funciones de utilidades realistas; imaginemos, por ejemplo, una red compleja de preferencias interactuantes que debe ser entendida por un agente que funciona como un asistente de oficina para un ser humano. Resulta muy difícil descomponer las preferencias en estados complejos de la misma forma que las redes de Bayes descomponen las creencias sobre los estados complejos. Una razón puede ser que las preferencias sobre los estados se compilan desde las preferencias sobre las historias de estados descritas por las **funciones de recompensa** (*véase* el Capítulo 17). Aunque la función de recompensa sea simple, la función de utilidad correspondiente puede ser muy compleja. Esto sugiere que hay que tomar muy en serio la tarea de ingeniería del conocimiento para las funciones de recompensa como una forma de transmitir a nuestros agentes qué es lo que queremos que hagan.

Aprendizaje: los Capítulos 18 y 20 describen la forma en que se puede formular el aprendizaje en un agente como aprendizaje inductivo (supervisado, sin supervisar o basado en el refuerzo) de las funciones que constituyen los diferentes componentes del agente. Se han desarrollado técnicas estadísticas y lógicas muy poderosas que pueden hacer frente a problemas bastante grandes, alcanzando o excediendo las capacidades

humanas en la identificación de patrones predictivos definidos en un vocabulario dado. Por otro lado, el aprendizaje de la máquina no ha avanzado mucho en cuanto al importante problema de construir representaciones nuevas a niveles de abstracción mayores que el vocabulario de entrada. Por ejemplo, ¿cómo puede generar un robot autónomo predicados útiles tales como *Oficina* y *Café* si no son proporcionados por los humanos? Consideraciones similares se aplican al comportamiento del aprendizaje, *TomarUnaTazaDeTé* es una acción de alto nivel importante, pero, ¿cómo introducirla en una biblioteca de acciones que inicialmente contiene acciones mucho más simples tales como *LevanteelBrazo* y *Trague*? A menos que entendamos estos temas, nos enfrentamos a la tarea desoladora de construir a mano grandes bases de conocimiento de sentido común.

27.2 Arquitecturas de agentes

Es natural que preguntemos, «¿Cuál de las arquitecturas de agentes del Capítulo 2 deberían utilizar los agentes?» La respuesta es «¡Todas ellas!». Hemos visto que para situaciones en las que el tiempo no es esencial, se necesitan respuestas reflejas, mientras que la deliberación basada en el conocimiento permite que el agente planifique con antelación. Un agente completo debe ser capaz de hacer las dos cosas, utilizando una **arquitectura híbrida**. Una propiedad importante de las arquitecturas híbridas es que los límites entre los diferentes componentes de decisión no son fijos. Por ejemplo, la **compilación**, a nivel deliberativo, convierte continuamente la información declarativa en representaciones más eficientes, alcanzando finalmente el nivel reflejo, *véase* la Figura 27.2 (Este es el propósito del aprendizaje basado en explicaciones que se examinó en el Capítulo 19). Las arquitecturas de agentes tales como SOAR (Laird *et al*., 1987) y THEO (Mitchell, 1990) tienen exactamente esta estructura. Cada vez que resuelven un problema mediante deliberación explícita se almacena una versión generalizada de la solución para usarla en un componente reflejo. Un problema menos abordado es la *inversión* de este proceso: cuando el entorno cambia, los reflejos aprendidos ya no son apropiados y el agente debe retornar al nivel deliberativo para producir comportamientos nuevos.

ARQUITECTURA HÍBRIDA

Figura 27.2 La compilación sirve para convertir la toma de decisión deliberativa en mecanismos reflexivos más eficientes.

Los agentes también necesitan formas de controlar sus propias deliberaciones. Deben poder parar de deliberar cuando se exige la acción, y deben poder utilizar el tiempo del que disponen para la deliberación y ejecutar los cómputos más rentables. Por ejemplo, un agente que conduce un taxi y ve un accidente delante, debe decidir en menos de un segundo si frenar o actuar de forma evasiva. También debería pensar en menos de un segundo las cuestiones más importantes, tales como si los carriles de la izquierda y derecha están libres y si tiene un camión detrás muy cerca, en vez de preocuparse de la tensión y el desgaste en las ruedas (neumáticos, gomas) o de dónde recoger al siguiente pasajero. Estos temas normalmente se estudian bajo el encabezamiento de **IA en tiempo real**. A medida que los sistemas IA entran en dominios más complejos, todos los problemas serán de tiempo real, porque el agente nunca tendrá tiempo suficiente como para resolver el problema de la decisión de forma exacta.

IA EN TIEMPO REAL

Obviamente, existe una necesidad urgente de métodos que funcionen en situaciones de toma de decisiones más generales. En los últimos años han surgido dos técnicas prometedoras. La primera conlleva la utilización de **algoritmos de cualquier momento** *(anytime)* (Dean y Boddy, 1988; Horvitz, 1987). Un algoritmo de esta clase es un algoritmo cuya calidad de salida mejora gradualmente con el tiempo, de manera que tiene preparada una decisión razonable siempre que tenga una interrupción. Dichos algoritmos se controlan mediante un procedimiento de decisiones de metanivel que evalúa si merece la pena realizar más cómputos. Una búsqueda iterativa cada vez más en profundidad en los juegos es un ejemplo simple de un algoritmo de cualquier momento. También se pueden construir sistemas más complejos, que se compongan de muchos algoritmos de esta clase trabajando juntos (Zilberstein y Russell, 1996). La segunda técnica es el **metarazonamiento teórico para las decisiones** (Horvitz, 1989; Russell y Wefald, 1991; Horvitz y Breese, 1996). Este método aplica la teoría del valor de la información (véase el Capítulo 16) para la selección de cómputos. El valor de un cómputo depende tanto de sus costes (en lo que se refiere a demorar la acción) como de sus beneficios (en cuanto a la calidad de decisión mejorada). Las técnicas del meta-razonamiento se pueden utilizar para diseñar mejores algoritmos de búsqueda y para garantizar que los algoritmos tienen la propiedad «cualquier momento» (*anytime*). Desde luego que el meta-razonamiento es caro y que los métodos de compilación se pueden aplicar para que los sobrecostes sean pequeños en comparación con los costes de los cómputos que se están controlando.

ALGORITMOS DE CUALQUIER MOMENTO

METARAZONAMIENTO TEÓRICO PARA LAS DECISIONES

El meta-razonamiento no es sino un aspecto de una **arquitectura reflexiva** general, es decir, una arquitectura que permite la deliberación sobre las entidades y las acciones computacionales que ocurren dentro de la misma arquitectura. Un fundamento teórico de las arquitecturas reflexivas se puede construir mediante la definición de un espacio de estados conjunto compuesto por el estado del entorno y del estado computacional del mismo agente. Se pueden diseñar algoritmos de aprendizaje y de toma de decisiones que funcionen en este espacio de estados conjunto y que por tanto sirvan para implementar y mejorar las actividades computacionales del agente. Finalmente, se espera que los algoritmos específicos para unas tareas tales como la búsqueda alfa-beta y el encadenamiento hacia atrás (*backward)* desaparezcan de los sistemas IA, para ser sustituidos por métodos generales que dirijan los cómputos del agente hacia la generación eficiente de decisiones de alta calidad.

ARQUITECTURA REFLEXIVA

27.3 ¿Estamos llevando la dirección adecuada?

En la sección anterior mencionamos muchos avances y oportunidades para progresar. Pero, ¿a qué nos lleva todo eso? Dreyfus (1992) ofrece la analogía de intentar llegar a la luna trepando un árbol, de manera que se puede informar de un progreso continuo hasta llegar a conseguir la copa del árbol. En esta sección, estudiaremos si el camino actual de la IA es más parecido a trepar un árbol o a un viaje en cohete.

En el Capítulo 1 dijimos que nuestro objetivo era construir agentes que actuaran racionalmente. Sin embargo, también dijimos que

> …alcanzar la racionalidad perfecta, haciendo siempre lo correcto, no es viable en entornos complicados. Las exigencias computacionales son simplemente muy elevadas. Sin embargo, en la mayor parte del libro, adoptaremos la hipótesis de trabajo en que la racionalidad perfecta es un buen punto de comienzo (arranque) para el análisis.

Ahora, es momento de considerar una vez más cuál es exactamente el objetivo de la IA. Queremos construir agentes, pero, ¿qué especificación debemos tener en mente? Existen cuatro posibilidades:

RACIONALIDAD PERFECTA

Racionalidad perfecta. Un agente perfectamente racional actúa en cualquier instante de tal manera que maximiza la utilidad esperada, dada la información que haya adquirido del entorno. Hemos observado que los cálculos necesarios para lograr la racionalidad perfecta en la mayoría de los entornos llevan demasiado tiempo, de forma que la racionalidad perfecta no es un objetivo realista.

RACIONALIDAD CALCULADORA

Racionalidad calculadora. Esta es la noción de la racionalidad que hemos utilizado implícitamente al diseñar agentes lógicos y teóricos para las decisiones. Un agente calculadoramente racional *finalmente* devuelve lo que *habría sido* la opción racional al comienzo de su deliberación. Esta es una propiedad interesante para que un sistema la presente; sin embargo en la mayoría de los entornos, la respuesta adecuada no tiene valor en el momento equivocado. En la práctica, los diseñadores de sistemas IA se ven obligados a llegar a un compromiso sobre la calidad de las decisiones para obtener un rendimiento global razonable; desgraciadamente, la base teórica de la racionalidad calculadora no proporciona un modo bien fundado para llegar a dichos compromisos.

RACIONALIDAD LIMITADA

Racionalidad limitada. Herbert Simon (1957) rechazó la noción de racionalidad perfecta (o incluso aproximadamente perfecta) y la reemplazó con la racionalidad limitada, una teoría descriptiva de la toma de decisiones por agentes reales. Y en sus propias palabras, él escribió:

> La capacidad de la mente humana para formular y resolver problemas complejos es muy pequeña en comparación con el tamaño de los problemas cuya solución se requiere para un comportamiento objetivamente racional en el mundo real, o incluso para una aproximación razonable a dicha racionalidad objetiva.

Simon sugirió que la racionalidad limitada funciona en primer lugar **satisfaciendo**, es decir, deliberando sólo el tiempo necesario para elaborar una respuesta que sea lo «suficientemente buena». Simon ganó el premio Nobel en economía por este trabajo y ha escrito mucho en profundidad acerca de este tema (Simon, 1982). En muchos casos, parece que es un modelo útil para los comportamientos humanos. Sin embargo, no es una

especificación formal para los agentes inteligentes, porque la definición de «suficientemente bueno» no la ofrece esta teoría. Además, satisfacer parece ser sólo uno de la gran cantidad de métodos que se utilizan para hacer frente a recursos limitados.

OPTIMALIDAD LIMITADA

Optimalidad limitada (*Bounded optimality,* BO). Un agente óptimo limitado se comporta todo lo bien que puede, *dado sus recursos computacionales*. Es decir, la utilidad esperada del programa agente para un agente óptimo limitado es por lo menos tan elevada como la utilidad esperada de cualquier otro programa agente que se ejecute en la misma máquina.

De estas cuatro posibilidades, la optimalidad limitada parece ofrecer la mejor esperanza de conseguir un fundamento teórico fuerte para la IA. Tiene la ventaja de que es posible conseguirla; existe siempre por lo menos un mejor programa, algo de lo que carece la racionalidad perfecta. Los agentes óptimos limitados son realmente útiles en el mundo real, mientras que los agentes calculadoramente racionales normalmente no lo son, y los agentes satisfactores podrían serlo o no dependiendo de sus propios caprichos.

El enfoque tradicional de la IA ha sido empezar con la racionalidad calculadora y a continuación hacer concesiones para satisfacer las restricciones de los recursos. Si los problemas impuestos por las restricciones son de menor importancia, se podría esperar que el diseño final sea similar al diseño de agentes BO. Pero a medida que las restricciones de los recursos se hacen más críticas, por ejemplo, a medida que el entorno es más complejo se podría esperar que los diseños vayan a divergir. En la teoría de la optimalidad limitada, estas restricciones se pueden abordar de una manera escrupulosa.

Hasta ahora, no se conoce mucho sobre la optimalidad limitada. Es posible construir programas de optimalidad limitada para máquinas muy simples y para clases de entornos algo restringidos (Etzioni, 1989; Russell *et al*., 1993), pero todavía no tenemos idea de cómo serán los programas BO para grandes computadores de uso general en entornos complejos. Si va a existir una teoría constructiva de la optimalidad limitada, tenemos que esperar que el diseño de los programas óptimos limitados no dependa demasiado de los detalles del computador que se está utilizando. Ello haría que la investigación científica fuera muy difícil si añadir unos pocos kilobytes de memoria a una máquina de gigabytes cambiara significativamente el diseño del programa BO. Una forma de asegurarnos de que esto no ocurra es estar ligeramente más tranquilos con los criterios de optimalidad limitada. Por analogía con la noción de complejidad asintótica (ABO)(Apéndice A), la **optimalidad limitada asintótica** (ABO, *Asymptotic Bounded Optimality*) se puede definir de la siguiente forma (Russell y Subramanian, 1995). Supongamos que el programa P es un programa óptimo limitado para la máquina M en la clase de entornos **E**, en donde la complejidad de los entornos en **E** no está limitada. Entonces el programa P' es ABO para M en **E** si puede superar a P ejecutándose en una máquina kM que es k veces más rápida (o mayor) que M. A menos que k fuera enorme, estaríamos encantados de que un programa fuera ABO para un entorno no trivial en una arquitectura no trivial. No tendría sentido esforzarnos por encontrar programas BO en vez de ABO, porque de todas formas el tamaño y la velocidad de las máquinas disponibles tiende a multiplicarse por un factor constante en una cantidad fija de tiempo.

OPTIMALIDAD ASINTÓTICA

Podríamos arriesgar el pronóstico de que los programas BO o ABO para computadores potentes en entornos complejos no tendrán necesariamente una estructura simple y elegante. Y hemos visto ya que la inteligencia de propósito general requiere capaci-

dad de reflejos y alguna capacidad deliberativa, una variedad de formas de conocimiento y toma de decisiones, mecanismos de aprendizaje y de compilación para todas esas formas, métodos de controlar el razonamiento, y un gran almacén de conocimientos específicos del dominio. Un agente óptimo limitado se debe adaptar al entorno en el que se encuentra de manera que su organización interna pueda reflejar las optimizaciones que sean específicas para ese entorno en particular. Esto es esperable, y es similar a la manera en que los coches de carreras restringidos por la capacidad del motor han evolucionado hasta tener diseños extremadamente complejos. Sospechamos que una ciencia de inteligencia artificial basada en la optimalidad limitada conllevará realizar un estudio amplio de los procesos que permiten a un programa de agentes converger hacia la optimalidad limitada y quizá con una concentración inferior en los detalles de los programas complicados que se obtengan como resultado.

Resumiendo, se propone un concepto de optimalidad limitada como una tarea formal para la investigación de la IA que esté bien definida y que sea viable. La optimalidad limitada especifica *programas* óptimos en vez de *acciones* óptimas. Las acciones, después de todo, son generadas por programas, y es sobre los programas en donde tienen control los diseñadores.

27.4 ¿Qué ocurriría si la IA tuviera éxito?

En la novela de David Lodge, *Small World* (1984) sobre el mundo académico de la crítica literaria, el protagonista causa consternación cuando pregunta a un panel de teóricos literarios eminentes, pero contradictorios la siguiente cuestión: «¿Qué ocurriría si ustedes tuvieran razón?» Ninguno de los teóricos parece haber considerado antes esta cuestión, quizá porque debatir teorías infalsificables es un fin en sí mismo. Una confusión similar algunas veces se puede evocar preguntando a los investigadores en IA, «¿Qué ocurriría si tuvieran éxito?». La IA es fascinante, los computadores inteligentes son obviamente más útiles que los computadores no inteligentes, por tanto ¿por qué hay que preocuparse?

Como se dijo en el Apartado 26.3, existen temas éticos a tener en cuenta. Los computadores inteligentes son más potentes, pero ¿se va a utilizar ese poder para fines buenos o malos? Aquellos que se esfuerzan por desarrollar la IA tienen la responsabilidad de ver que el impacto de su trabajo es positivo. El alcance de este impacto dependerá del grado de éxito de la IA. Y ya se han obtenido éxitos modestos que han cambiado las formas de enseñar la informática (Stein, 2002) y las formas en que se practica el desarrollo del *software*. La IA ha hecho posibles aplicaciones nuevas tales como los sistemas de reconocimiento de voz, sistemas de control de inventarios, sistemas de vigilancia, robots y motores de búsqueda.

Se puede esperar que los éxitos de medio nivel en IA afectarán a toda clase de gente en sus vidas cotidianas. Hasta ahora, las redes de comunicaciones computerizadas, tales como los teléfonos móviles (celulares) e Internet, han tenido este tipo de efecto penetrante o dominante en la sociedad, pero la IA no. Podemos imaginar que los asistentes personales verdaderamente útiles para la oficina o para casa tendrían un impacto

positivo sobre las vidas de las personas, aunque a corto plazo podrían causar dislocaciones económicas. Una capacidad tecnológica a este nivel también se puede aplicar al desarrollo de armas autónomas, que muchos ven como un avance indeseable.

Finalmente, parece probable que un éxito en IA a gran escala, la creación de inteligencia en el nivel humano y más allá, cambiaría las vidas a una mayoría de la humanidad. La verdadera naturaleza de nuestro trabajo y de nuestro papel cambiaría así como nuestro punto de vista de la inteligencia, la consciencia y el destino futuro de la raza humana. A este nivel, los sistemas IA podrían suponer una amenaza directa a la autonomía humana, la libertad, e incluso la supervivencia. Por estas razones, la investigación en IA no se puede divorciar de sus consecuencias éticas.

¿Qué camino deparará el futuro? Los autores de ciencia ficción parecen estar a favor de futuros anti-utópicos en vez de los utópicos, probablemente porque se producen argumentos más interesantes. Sin embargo hasta la fecha, la IA parece encajar con otras tecnologías revolucionarias (imprenta, fontanería, viajes en avión, telefonía) cuyas repercusiones negativas son compensadas por sus aspectos positivos.

En conclusión, se puede observar que la IA ha progresado a lo largo de su corta existencia, pero todavía hoy tiene validez la frase del ensayo de Alan Turing sobre *Computing Machinery and Intelligence*:

> «Podemos observar sólo a poca distancia hacia delante, pero también se puede ver que todavía queda mucho por hacer.»

A Fundamentos matemáticos

A.1 Análisis de la complejidad y la notación O( )

PRUEBAS DE EVALUACIÓN

Los científicos informáticos se suelen enfrentar con la tarea de comparar algoritmos para ver la rapidez de ejecución y la cantidad de memoria que requieren. Existen dos enfoques para abordar esta tarea. El primero son las **pruebas de evaluación** (***benchmarking***), ejecutar los algoritmos en un computador y medir la velocidad en segundos y el consumo de memoria en bytes. En última instancia, esto es lo que importa de verdad, pero una prueba de evaluación puede no resultar satisfactoria porque es tan específica: mide el rendimiento de un programa en particular escrito en un lenguaje de programación, ejecutándose en un computador específico, con un compilador específico y una entrada de datos específica. A partir del resultado que se obtenga de esta prueba puede resultar difícil predecir el funcionamiento del algoritmo en un compilador, computador o conjunto de datos diferentes.

Análisis asintótico

ANÁLISIS DE ALGORITMOS

El segundo enfoque depende del **análisis de algoritmos**, independientemente de la implementación y entrada en particular. Examinaremos este enfoque mediante el ejemplo siguiente, un programa para calcular la suma de una secuencia de números:

```
función SUMA(secuencia) devuelve un número
    suma ← 0
    desde i ← 1 hasta LONGITD(secuencia)
        suma ← suma + secuencia[i]
    devolver suma
```

El primer paso del análisis es abstraer la entrada, para encontrar algún parámetro o parámetros que caractericen el tamaño de la entrada. En este ejemplo, la entrada puede ser caracterizada por la longitud de la secuencia, que llamaremos n. El segundo paso es abstraer la implementación, para encontrar alguna medida que refleje el tiempo de ejecución del algoritmo, pero que no esté ligado a un compilador o computador en particular. Para el programa SUMA, esto podría ser simplemente el número de líneas de código ejecutadas, o podría ser más detallado midiendo el número de sumas, asignaciones, referencias a arrays y bifurcaciones ejecutadas por el algoritmo. Ambas formas ofrecen una caracterización del número total de pasos del algoritmo como una función del tamaño de la entrada. A esta caracterización la llamaremos $T(n)$. Si contamos las líneas de código, para este ejemplo tendremos $T(n) = 2n + 2$.

Si todos los programas fueran tan simples como el programa SUMA, el análisis de los algoritmos sería un campo trivial. En primer lugar, es extraño encontrar un parámetro como n que caracterice completamente el número de pasos realizados por el algoritmo. En lugar de eso, lo mejor que podemos hacer es calcular el peor caso $T_{\text{peor}}(n)$ o el caso promedio $T_{\text{prom}}(n)$. Computar un promedio significa que el analista presupone alguna distribución de las entradas.

El segundo problema es que los algoritmos tienden a resistirse al análisis exacto. En ese caso, es necesario replegarse a una aproximación. Decimos que el algoritmo SUMA es $O(n)$, lo que significa que su medida es a lo sumo una constante por n, con la posible excepción de unos cuantos valores pequeños de n. Más formalmente,

$$T(n) \text{ es } O(f(n)) \text{ si } T(n) \leqslant k\,f(n) \text{ para algunos } k, \text{ para todo } n > n_0.$$

ANÁLISIS ASINTÓTICO

La notación $O(\ )$ nos proporciona lo que llamamos un **análisis asintótico**. Sin lugar a dudas, podremos decir que, como n se aproxima asintóticamente al infinito, un algoritmo $O(\ )$ es mejor que un algoritmo $O(n^2)$. Una sola cifra de prueba de evaluación no podría corroborar dicha afirmación.

La notación $O(\ )$ abstrae los factores constantes, lo cual facilita su utilización, aunque de forma menos precisa, que la notación $T(\ )$. Por ejemplo, un algoritmo $O(n^2)$ a la larga siempre será peor que uno $O(n)$, pero si los dos algoritmos son $T(n^2 + 1)$ y $T(100n + 1000)$, entonces el algoritmo $O(n^2)$ es realmente mejor para $n = 110$.

A pesar de este inconveniente, el análisis asintótico es la herramienta más utilizada para analizar algoritmos. Es así precisamente porque el análisis abstrae tanto el número exacto de operaciones (ignorando el factor constante k) como el contenido exacto de la entrada (teniendo en cuenta sólo el tamaño n) que se convierte en matemáticamente factible. La notación $O(\ )$ es un buen compromiso entre la precisión y la facilidad de análisis.

Los problemas inherentemente difíciles y NP

El análisis de algoritmos y la notación $O(\ )$ nos permiten hablar sobre la eficiencia de un algoritmo en particular. Sin embargo, no tienen nada que decir con respecto a si podría existir un algoritmo mejor para el problema actual. El campo del **análisis de complejidad** analiza problemas, no algoritmos. La primera gran división se realiza entre los problemas que se pueden resolver en tiempo polinomial y los problemas que no pueden

ANÁLISIS DE COMPLEJIDAD

resolverse en tiempo polimonial, sin importar el algoritmo que se use. La clase de los problemas polinomiales, los que se pueden resolver en tiempo $O(n^k)$ para k, se llama P. Algunas veces se llaman problemas «fáciles», porque la clase contiene esos problemas con tiempo $O(\log n)$ y $O(n)$. Sin embargo también contiene los problemas con tiempos de ejecución como $O(n^{1000})$, de manera que la denominación de «fácil» no se debería tomar demasiado al pie de la letra.

Otra clase importante de problemas es NP, la clase de problemas polinomiales no deterministas. Un problema se encuentra en esta clase si existe algún algoritmo que pueda averiguar una solución y a continuación verificar si esa averiguación es correcta en tiempo polinomial. La idea es que si tenemos un número de procesadores arbitrariamente grande, para probar todas las averiguaciones de una vez, o si somos afortunados y siempre averiguamos lo correcto la primera vez, entonces los problemas NP se convierten en problemas P. Una de las preguntas más importantes y más abiertas en informática es si la clase NP equivale a la clase P cuando no se tiene el lujo de tener un número infinito de procesadores o averiguaciones omniscientes. La mayoría de los informáticos están convencidos de que P $\neq$ NP, es decir que los problemas NP son inherentemente difíciles y no tienen algoritmos de tiempo polinomial. Pero esto todavía no se ha podido demostrar.

COMPLETOS NP

Todo el que esté interesado en decidir si P = NP, tendrá que estudiar una subclase de NP llamada problemas **completos NP**. La palabra «completos» se utiliza en el sentido de «los más extremos» y se refiere por tanto a los problemas más difíciles de la clase NP. Se ha demostrado que o están en P todos los problemas completos de NP o no está ninguno. Esto hace que esta clase sea teóricamente interesante, pero la clase también es de interés práctico, porque muchos problemas importantes se sabe que son problemas NP completos. Un ejemplo es el problema de la satisfabilidad: dada una sentencia de lógica proposicional, ¿existe una asignación de valores verdad a los símbolos de proposición de la sentencia que hagan que sea verdadera? A menos que ocurra un milagro y P = NP, no puede existir un algoritmo que resuelva *todos* los problemas en tiempo polinomial. Sin embargo, la IA está mucho más interesada en si existen algoritmos que funcionen más eficientemente con problemas *típicos* extraídos de una distribución predeterminada; como vimos en el Capítulo 7, existen algoritmos tales como WALKSAT, que funcionan muy bien en muchos problemas.

CO-NP

La clase **co-NP** es el complemento de NP, en el sentido de que para todos los problemas de decisiones en NP, existe un problema correspondiente en co-NP con las respuestas «si» y «no» invertidas. Sabemos que P es un subconjunto tanto de co-NP como de NP, y se cree que hay problemas en co-NP que no están en P. Los problemas **completos co-NP** son los más difíciles de co-NP.

COMPLETOS CO-NP

La clase #P (pronunciado *«sharp P», «P almohadilla»*) es el conjunto de problemas de cómputo correspondiente a los problemas de decisiones en NP. Los problemas de decisiones obtienen la respuesta sí-o-no: ¿existe una solución para la fórmula 3-SAT? Los problemas de cómputo obtienen como respuesta un entero: ¿cuántas soluciones existen para la fórmula 3-SAT? En algunos casos, el problema del cómputo es mucho más difícil que el problema de la decisión. Por ejemplo, decidir si un grafo bipartito tiene una correspondencia perfecta se puede hacer en tiempo $O(V\,E)$ (en donde el grafo tiene V vértices y E bordes), sin embargo el problema de cálculos «cuántas correspondencias per-

fectas tiene este grafo bipartito» es un problema completo de #P y es por lo menos tan difícil como cualquier problema NP.

También se ha estudiado la clase de problemas PSPACE, los que requieren una cantidad de espacio polinomial, incluso en una máquina no determinista. Se piensa que los problemas PSPACE difíciles son peores que los problemas completos NP, aunque podría resultar que NP = PSPACE, de la misma forma que podría ser P = NP.

A.2 Vectores, matrices y álgebra lineal

VECTOR

Los matemáticos definen un **vector** como un miembro de un espacio vectorial, sin embargo nosotros vamos a utilizar una definición mucho más concreta: un vector es una secuencia ordenada de valores. Por ejemplo, en el espacio bi-dimensional, tenemos dos vectores tales como $\mathbf{x} = \langle 3, 4 \rangle$ e $\mathbf{y} = \langle 0, 2 \rangle$. Nosotros seguimos la convención usual de utilizar caracteres en negrita para los nombres de los vectores, aunque algunos autores utilicen las flechas o las barras sobre los nombres: $\vec{x}$ o $\overline{y}$. Se puede acceder a los elementos de un vector mediante subíndices: $\mathbf{z} = \langle z_1, z_2, \ldots, z_n \rangle$.

Las dos operaciones fundamentales de los vectores son la suma de vectores y la multiplicación escalar. La suma de vectores $\mathbf{x} + \mathbf{y}$ es la suma elemento a elemento: $\mathbf{x} + \mathbf{y} = \langle 3 + 0, 4 + 2 \rangle = \langle 3, 6 \rangle$. La multiplicación escalar multiplica cada elemento por una constante: $5\mathrm{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$.

La longitud de un vector se denota $|\mathbf{x}|$ y se calcula tomando la raíz cuadrada de la suma de los cuadrados de los elementos: $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$. El producto punto (también llamado producto escalar) de dos vectores $\mathbf{x} \cdot \mathbf{y}$ es la suma de los productos de los elementos correspondientes, es decir, $\mathbf{x} \cdot \mathbf{y} = \Sigma_i\, x_i y_i$, o en nuestro caso particular, $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$.

Los vectores se suelen interpretar como segmentos de líneas dirigidas (flechas) en un espacio n-dimensional. La suma de vectores entonces es equivalente a colocar la cola del vector al principio del otro, y el producto punto $\mathbf{x} \cdot \mathbf{y}$ es igual a $|\mathbf{x}|\,|\mathbf{y}| \cos \theta$, en donde θ es el ángulo entre $\mathbf{x}$ e $\mathbf{y}$.

MATRIZ

Una **matriz** es un array rectangular de valores organizada en filas y columnas. Esta es la matriz $\mathbf{m}$ de tamaño 3×4.

$$\begin{pmatrix} \mathbf{m}_{1,1} & \mathbf{m}_{1,2} & \mathbf{m}_{1,3} & \mathbf{m}_{1,4} \\ \mathbf{m}_{2,1} & \mathbf{m}_{2,2} & \mathbf{m}_{2,3} & \mathbf{m}_{2,4} \\ \mathbf{m}_{3,1} & \mathbf{m}_{3,2} & \mathbf{m}_{3,3} & \mathbf{m}_{3,4} \end{pmatrix}$$

El primer índice de $\mathbf{m}_{i,j}$ especifica la fila y el segundo la columna. En los lenguajes de programación, $\mathbf{m}_{i,j}$ a menudo se escribe `m[i,j]` o `m[i][j]`.

La suma de las dos matrices se define sumando los elementos correspondientes; así pues, $(\mathbf{m} + \mathbf{n})_{i,j} = \mathbf{m}_{i,j} + \mathbf{n}_{i,j}$. (La suma no se define si $\mathbf{m}$ y $\mathbf{n}$ tienen diferentes tamaños). También podemos definir la multiplicación de una matriz por un escalar: $(c\mathbf{m})_{i,j} = c\mathbf{m}_{i,j}$. La multiplicación de matrices (el producto de dos matrices) es más complicada. El producto $\mathbf{mn}$ se define sólo si $\mathbf{m}$ es de tamaño $a \times b$ y $\mathbf{n}$ de tamaño $b \times c$ (es decir, la segunda matriz tiene el mismo número de filas que la primera de columnas); el resultado

es una matriz de tamaño $a \times c$. Esto significa que la multiplicación de matrices no es conmutativa: $\mathbf{mn} \neq \mathbf{mn}$ en general. Si las matrices son del tamaño apropiado, entonces el resultado es

$$(\mathbf{mn})_{i,k} = \sum_i \mathbf{m}_{i,j}\,\mathbf{n}_{j,k}$$

La matriz de identidad $\mathbf{I}$ tiene elementos $\mathbf{I}_{i,j}$ iguales a 1 cuando $i = j$ e iguales a 0 en otros casos. Tiene la propiedad de que $\mathbf{mI} = \mathbf{m}$ para toda $\mathbf{m}$. La transposición de $\mathbf{m}$, escrito $\mathbf{m}^T$ se forma convirtiendo filas en columnas y viceversa, o de manera más formal, mediante $\mathbf{m}^T_{i,j} = \mathbf{m}_{j,i}$.

ELIMINACIÓN GAUSS-JORDAN

Las matrices se utilizan para resolver sistemas de ecuaciones lineales mediante un proceso llamado **eliminación Gauss-Jordan**, un algoritmo $O(n^3)$. Consideremos el siguiente grupo de ecuaciones, para el que queremos una solución en x, y y z:

$$\begin{aligned} +2x + y - z &= 8 \\ -3x - y + 2z &= 11 \\ -2x + y + 2z &= -3. \end{aligned}$$

Podemos representar este sistema como una matriz:

$$\begin{array}{c} \begin{array}{cccc} x & y & z & c \end{array} \\ \begin{pmatrix} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{pmatrix} \end{array}$$

Aquí, $x\ y\ z\ c$ no es parte de la matriz; existe simplemente un recordatorio para el que lo ve. Sabemos que si multiplicamos ambos lados de una ecuación por una constante o sumamos dos ecuaciones, obtenemos una ecuación igualmente válida. La eliminación Gauss-Jordan funciona realizando estas operaciones de forma repetida de tal forma que comenzamos eliminando la primera variable (x) de todo, excepto de la primera ecuación, y entonces continuamos eliminando la variable i-ésima de todo excepto de la ecuación i-ésima, para todo i. Eliminamos x de la segunda ecuación multiplicando la primera ecuación por 3/2 y sumándola a la segunda. Esto da la matriz siguiente:

$$\begin{array}{c} \begin{array}{cccc} x & y & z & c \end{array} \\ \begin{pmatrix} 2 & 1 & -1 & 8 \\ 0 & 0{,}5 & 0{,}5 & 1 \\ -2 & 1 & 2 & -3 \end{pmatrix} \end{array}$$

Continuamos de esta manera, eliminando x, y y z hasta obtener

$$\begin{array}{c} \begin{array}{cccc} x & y & z & c \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{pmatrix} \end{array}$$

indicando que $x = 2$, $y = 3$, $z = -1$ es una solución. (¡Inténtelo!)

A.3 Distribuciones de probabilidades

Una probabilidad es una medida sobre un conjunto de sucesos (eventos) que satisface tres axiomas:

1. La medida de cada suceso está entre 0 y 1. Escribimos esto como $0 \leq P(E = e_i) \leq 1$, en donde E es una variable aleatoria que representa un evento y e_i son valores posibles de E. En general, las variables aleatorias se denotan por letras mayúsculas y sus valores por letras minúsculas.
2. La medida del conjunto completo es 1; es decir $\Sigma_{i=1}^{n} P(E = e_i) = 1$.
3. La probabilidad de una unión de sucesos disjuntos es la suma de las probabilidades de los sucesos individuales; es decir, $P(E = e_1 \vee E = e_2) = P(E = e_1) + P(E = e_2)$, donde e_1 y e_2 son disjuntos.

Un **modelo probabilístico** consiste en un espacio muestral de resultados posibles mutuamente excluyentes, junto con una medida de probabilidad para cada resultado. Por ejemplo, en un modelo de pronóstico del tiempo para mañana, los resultados podrían ser *soleado*, *nubloso*, *lluvioso* y *con nieve*. Un subconjunto de estos resultados constituye un suceso. Por ejemplo, el evento de precipitaciones es el conjunto {*lluvioso*, *con nieve*}.

Utilizaremos $\mathbf{P}(E)$ para denotar el vector de valores $\langle P(E = e_1), \ldots, P(E = e_n)\rangle$. También utilizamos $P(e_i)$ como una abreviatura para $P(E = e_1)$ y $\Sigma_e \mathrm{P}(e)$ para $\Sigma_{i=1}^{n} P(E = e_i)$.

La probabilidad condicional $P(B|A)$ se define como $P(B \cap A)/P(A)$. A y B son independientes condicionalmente si $P(B|A) = P(B)$ (o de manera equivalente, $P(B|A) = P(A)$). Para las variables continuas, existe un número infinito de valores, y a menos que existan picos, la probabilidad de cualquier valor es 0. Por tanto, definimos una **función de densidad de probabilidad**, que denotamos también como $P(X)$, pero que tiene un significado ligeramente diferente de la función de probabilidad discreta $P(A)$. La función de densidad $P(X = c)$ se define como el *ratio* de la probabilidad de que X se encuentra en un intervalo en torno a c, dividido por el ancho del intervalo, a medida que el ancho del intervalo tiende a cero:

FUNCIÓN DE DENSIDAD DE PROBABILIDAD

$$P(X = c) = \lim_{dx \to 0} P\,(c \leqslant X \leqslant c + dx)/dx\,.$$

La función de densidad debe ser no negativa para toda x y debe tener

$$\int_{-\infty}^{\infty} P(X)dx = 1.$$

También definimos una **función de densidad de probabilidades acumulada** $F(X)$, que es la probabilidad de que una variable aleatoria sea menor que x:

FUNCIÓN DE DENSIDAD DE PROBABILIDAD ACUMULADA

$$F(X) = \int_{-\infty}^{x} P(Z)dz.$$

Tengamos en cuenta que la función de densidad tiene unidades, mientras que la función de probabilidades discreta no tiene unidades. Por ejemplo, si X se mide en segundos,

entonces la densidad se mide en Hz (es decir, 1/seg.) Si **X** es un punto en el espacio tridimensional medido en metros, entonces la densidad se mide en $1/m^3$.

DISTRIBUCIÓN GAUSSIANA

Una de las distribuciones de probabilidad más importantes es la **distribución gaussiana**, también conocida como la **distribución normal**. Una distribución Gaussiana, con media μ y desviación estándar σ (y por tanto una varianza σ^2) se define como

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-(x-\mu)^2/(2\sigma^2)},$$

DISTRIBUCIÓN NORMAL ESTÁNDAR

en donde x es una variable continua que va desde $-\infty$ a $+\infty$. Con $\mu = 0$ medio y discrepancia $\sigma^2 = 1$, obtenemos el caso especial de la **distribución normal estándar**. Para una distribución sobre un vector **x** en d dimensiones, existe la **distribución gaussiana multivariada**:

DISTRIBUCIÓN GAUSSIANA MULTIVARIADA

$$P(\mathbf{x}) = \frac{1}{(2\pi)^n|\Sigma|}\, e^{-\frac{1}{2}((\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu))}$$

MATRIZ DE CO-VARIANZA

en donde μ es el vector media y Σ es la **matriz de co-varianza** de la distribución.

En una dimensión, también podemos definir la **distribución acumulada** $F(x)$ como la probabilidad de que una variable aleatoria sea menor que x. Para la distribución normal estándar, esto está dado como

$$F(X) = \int_{-\infty}^{x} P(x)dx = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right),$$

en donde $\mathrm{erf}(x)$ es la llamada **función de errores**, que no tiene representación formal cerrada.

TEOREMA DE LÍMITE CENTRAL

El **teorema de límite central** afirma que la media de n variables aleatorias tiende a una distribución normal a medida que n tiende al infinito. Esto se mantiene para cualquier grupo de variables aleatorias, a menos que la varianza de cualquier subconjunto finito de variables domine a las otras.

Notas bibliográficas e históricas

La notación $O(\,)$ que tanto se utiliza actualmente en informática fue introducida por primera vez en el contexto de una teoría de números por el matemático alemán P.G.H. Bachmann (1984). El concepto de NP-completitud fue inventado por Cook (1971), y el método moderno para establecer una reducción de un problema a otro se debe a Karp (1972). Cook y Karp han ganado el premio Turing, el mayor reconocimiento en informática, por su trabajo.

Entre los trabajos clásicos sobre análisis y diseño de algoritmos se incluyen los de Knuth (1973) y Aho, Hopcroft y Ullman (1974); contribuciones más recientes son las de Tarjan (1983) y Cormen, Leiserson y Rivest (1990). Estos libros ponen énfasis en el diseño y el análisis de algoritmos para resolver problemas tratables. Para la teoría de NP completitud y otras formas de intratabilidad, *véase* Garey y Johnson (1979) o Papadimitriou (1994). Además de esta teoría, Garey y Johnson proporcionan ejemplos

que representan con mucha fuerza por qué los científicos informáticos son unánimes al establecer la línea entre los problemas tratables e intratables en la frontera entre la complejidad de tiempo polinomial y exponencial. También proporcionan un catálogo voluminoso de problemas que son conocidos por ser completos NP o por otro lado intratables.

Entre otros textos interesantes sobre álgebra lineal se incluyen a Chung (1979) y Ross (1988). Para probabilidad, son valiosas las obras de Bertsekas y Tsitsiklis (2002) y Feller (1971).

B Notas sobre lenguajes y algoritmos

B.1 Definición de lenguajes con Backus-Naur Form (BNF)

En este libro, hemos definido varios lenguajes incluyendo los lenguajes de lógica proposicional (Apartado 7.4), lógica de primer orden (Apartado 8.2) y un subconjunto del inglés (Apartado 22.3). Un lenguaje formal se define como un conjunto de cadenas en donde cada cadena es una secuencia de símbolos. Todos los lenguajes en los que estamos interesados se componen de un conjunto infinito de cadenas, de manera que necesitamos una manera concisa de caracterizar ese conjunto. Eso es lo que hacemos con una **gramática**. Escribimos las gramáticas en un formalismo llamado **Backus-Naur Form (BNF)**. En una gramática BNF existen cuatro componentes:

BACKUS-NAUR FORM (BNF)

SÍMBOLOS TERMINALES

SÍMBOLOS NO TERMINALES

SÍMBOLOS DE INICIO

- Un conjunto de **símbolos terminales**. Estos son los símbolos o palabras que componen las cadenas del lenguaje. Podrían ser letras (**A**, **B**, **C**, …) o las palabras (**a**, **aardvark**, **abacus**, ...).
- Un conjunto de **símbolos no terminales** para *categorizar* (dividir por categorías) subfrases del lenguaje. Por ejemplo, el símbolo no terminal *FraseNominal* en inglés denota un conjunto infinito de cadenas incluyendo «you» y «the big slobbery dog» (el perro grande y baboso).
- El **símbolo de inicio** es un símbolo no terminal que representa las cadenas completas del lenguaje. En inglés, esto es una *Sentencia*; en aritmética podría ser *Expr*.
- Un conjunto de **reglas de reescritura**, de la forma $LHS \rightarrow RHS$, en donde *LHS* no es terminal y *RHS* es una secuencia de cero o más símbolos (o bien terminal o no terminal).

Una regla de reescritura de la forma

$$Sentencia \rightarrow FraseNominal\ FraseVerbal$$

significa que siempre que tengamos dos cadenas clasificadas como un *FraseNominal* y un *FraseVerbal*, podremos juntarlas y categorizar el resultado como una *Sentencia*. Como abreviatura, para separar lados derechos alternativos se puede utilizar el símbolo |. A continuación se muestra una gramática BNF para expresiones aritméticas sencillas:

$$\begin{array}{lcl} Expr & \rightarrow & Expr\ Operador\ Expr \mid (Expr) \mid Número \\ Número & \rightarrow & Dígito \mid Número\ Dígito \\ Dígito & \rightarrow & \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9} \\ Operador & \rightarrow & + \mid - \mid \div \mid \times \end{array}$$

En el Capítulo 22 los lenguajes y las gramáticas se abordaron con más profundidad. Somos conscientes de que otros libros utilizan notaciones ligeramente diferentes de BNF; por ejemplo, se puede utilizar ⟨*Dígito*⟩ en vez de *Dígito* para un no terminal, «palabra» en vez de **palabra** para un terminal, o bien : : = en vez de → en una regla.

B.2 Algoritmos de descripción en pseudocódigo

En este libro se definen con detalle más de 80 algoritmos. En lugar de seleccionar un lenguaje de programación (y correr el riesgo ante la posibilidad de que los lectores que no están familiarizados se pierdan), hemos optado por describir algoritmos en pseudocódigo. La mayor parte del pseudocódigo debería resultar familiar para usuarios de lenguajes como Java, C++ y Lisp. En algunos sitios utilizamos fórmulas matemáticas o inglés ordinario (o español) para describir partes que de otra manera serían muy pesadas. Deberían señalarse algunas idiosincrasias.

Variables estáticas. Utilizamos la palabra clave **estática** para decir que una variable recibe un valor inicial la primera vez que se llama a una función y retiene ese valor (o el valor asignado por la sentencia de asignación siguiente) en las siguientes llamadas a la función. Por consiguiente, las variables estáticas son como las variables globales que sobreviven a una única llamada a su función, pero son accesibles sólo dentro de la función. Los programas de agentes de este libro utilizan variables estáticas para la «memoria». Los programas con variables estáticas se pueden implementar como «objetos» en lenguajes orientados a objetos, tales como Java y Smalltalk. En los lenguajes funcionales, se pueden implementar por cierres funcionales en un entorno en el que se definen las variables requeridas.

Funciones como valores. Las funciones y los procedimientos tienen nombres en mayúsculas, y las variables tienen nombres en cursiva y minúscula. Por tanto la mayor parte del tiempo, una llamada a función se asemeja a FN(x). Sin embargo, permitimos

que el valor de una variable sea una función; por ejemplo, si el valor de la variable *f* es la función raíz cuadrada, entonces *f*(9) devuelve 3.

Los arrays comienzan en 1. A menos que se afirme otra cosa, el primer índice de un array es 1 como en la notación matemática usual, no 0, como en Java y en C.

La indentación (el sangrado) es significativa. La indentación se utiliza para marcar el ámbito (alcance) de un bucle o condicional, como en el lenguaje Python, y a diferencia de Java y C++ (que utilizan corchetes) o Pascal y Visual Basic que (utilizan **end**).

B.3 Ayuda en línea

La mayoría de los algoritmos de este libro se han implementado en nuestro *repositorio* (depósito) de código en línea:

aima.cs.berkeley.edu

Si tiene algún comentario, corrección o sugerencia para mejorar el libro, nos gustaría conocerlos. Por favor, visite la página Web para instrucciones y listas de discusión, o envíe un mensaje de correo electrónico a:

aima@cs.berkeley.edu

Bibliografia

Aarup, M., Arentoft, M. M., Parrod, Y., Stader, J., and Stokes, I. (1994). OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Fox, M. and Zweben, M. (Eds.), *Knowledge Based Scheduling*. Morgan Kaufmann, San Mateo, California.

Abramson, B. and Yung, M. (1989). Divide and conquer under global constraints: A solution to the N-queens problem. *Journal of Parallel and Distributed Computing*, *6*(3), 649–662.

Ackley, D. H. and Littman, M. L. (1991). Interactions between learning and evolution. In Langton, C., Taylor, C., Farmer, J. D., and Ramussen, S. (Eds.), *Artificial Life II*, pp. 487–509. Addison-Wesley, Redwood City, California.

Adelson-Velsky, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovsky, A. A., and Uskov, A. V. (1970). Programming a computer to play chess. *Russian Mathematical Surveys*, *25*, 221–262.

Adelson-Velsky, G. M., Arlazarov, V. L., and Donskoy, M. V. (1975). Some methods of controlling the tree search in chess programs. *Artificial Intelligence*, *6*(4), 361–371.

Agmon, S. (1954). The relaxation method for linear inequalities. *Canadian J. Math.*, *6*(3), 382–392.

Agre, P. E. and Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 268–272, Milan. Morgan Kaufmann.

Aho, A. V., Hopcroft, J., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts.

Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, Upper Saddle River, New Jersey.

Ait-Kaci, H. and Podelski, A. (1993). Towards a meaning of LIFE. *Journal of Logic Programming*, *16*(3–4), 195–234.

Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, *25*, 821–837.

Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, *97*, 270–277.

Aldous, D. and Vazirani, U. (1994). "Go with the winners" algorithms. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 492–501, Santa Fe, New Mexico. IEEE Computer Society Press.

Allais, M. (1953). Le comportment de l'homme rationnel devant la risque: critique des postulats et axiomes de l'école Américaine. *Econometrica*, *21*, 503–546.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery*, *26*(11), 832–843.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*, 123–154.

Allen, J. F. (1991). Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, *6*, 341–355.

Allen, J. F. (1995). *Natural Language Understanding*. Benjamin/Cummings, Redwood City; California.

Allen, J. F., Hendler, J., and Tate, A. (Eds.). (1990). *Readings in Planning*. Morgan Kaufmann, San Mateo, California.

Almuallim, H. and Dietterich, T. (1991). Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Vol. 2, pp. 547–552, Anaheim, California. AAAI Press.

ALPAC (1966). Language and machines: Computers in translation and linguistics. Tech. rep. 1416, The Automatic Language Processing Advisory Committee of the National Academy of Sciences, Washington, DC.

Alshawi, H. (Ed.). (1992). *The Core Language Engine*. MIT Press, Cambridge, Massachusetts.

Alterman, R. (1988). Adaptive planning. *Cognitive Science*, *12*, 393–422.

Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie, D. (Ed.), *Machine Intelligence 3*, Vol. 3, pp. 131–171. Elsevier/North-Holland, Amsterdam, London, New York.

Ambros-Ingerson, J. and Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pp. 735–740, St. Paul, Minnesota. Morgan Kaufmann.

Amit, D., Gutfreund, H., and Sompolinsky, H. (1985). Spin-glass models of neural networks. *Physical Review*, *A 32*, 1007–1018.

Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. (1989). HUGIN—a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 2, pp. 1080–1085, Detroit. Morgan Kaufmann.

Anderson, A. R. (Ed.). (1964). *Minds and Machines*. Prentice-Hall, Upper Saddle River, New Jersey.

Anderson, J. A. and Rosenfeld, E. (Eds.). (1988). *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, Massachusetts.

Anderson, J. R. (1980). *Cognitive Psychology and Its Implications*. W. H. Freeman, New York.

Anderson, J. R. (1983). *The Architecture of Cognition*. Harvard University Press, Cambridge, Massachusetts.

Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pp. 119–125, Edmonton, Alberta. AAAI Press.

Anshelevich, V. A. (2000). The game of Hex: An automatic theorem proving approach to game programming. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pp. 189–194, Austin, Texas. AAAI Press.

Anthony, M. and Bartlett, P. (1999). *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, Cambridge, UK.

Appel, K. and Haken, W. (1977). Every planar map is four colorable: Part I: Discharging. *Illinois J. Math.*, *21*, 429–490.

Apt, K. R. (1999). The essence of constraint propagation. *Theoretical Computer Science*, *221*(1–2), 179–210.

Apté, C., Damerau, F., and Weiss, S. (1994). Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, *12*, 233–251.

Arkin, R. (1998). *Behavior-Based Robotics*. MIT Press, Boston, MA.

Armstrong, D. M. (1968). *A Materialist Theory of the Mind*. Routledge and Kegan Paul, London.

Arnauld, A. (1662). *La logique, ou l'art de penser*. Chez Charles Savreux, au pied de la Tour de Nostre Dame, Paris.

Arora, S. (1998). Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the Association for Computing Machinery*, *45*(5), 753–782.

Ashby, W. R. (1940). Adaptiveness and equilibrium. *Journal of Mental Science*, *86*, 478–483.

Ashby, W. R. (1948). Design for a brain. *Electronic Engineering*, *December*, 379–383.

Ashby, W. R. (1952). *Design for a Brain*. Wiley, New York.

Asimov, I. (1942). Runaround. *Astounding Science Fiction*, *March*.

Asimov, I. (1950). *I, Robot*. Doubleday, Garden City, New York.

Astrom, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, *10*, 174–205.

Audi, R. (Ed.). (1999). *The Cambridge Dictionary of Philosophy*. Cambridge University Press, Cambridge, UK.

Austin, J. L. (1962). *How To Do Things with Words*. Harvard University Press, Cambridge, Massachusetts.

Axelrod, R. (1985). *The Evolution of Cooperation*. Basic Books, New York.

Bacchus, F. (1990). *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, Cambridge, Massachusetts.

Bacchus, F. and Grove, A. (1995). Graphical models for preference and utility. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, pp. 3–10, Montreal, Canada. Morgan Kaufmann.

Bacchus, F. and Grove, A. (1996). Utility independence in a qualitative decision theory. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning*, pp. 542–552, San Mateo, California. Morgan Kaufmann.

Bacchus, F., Grove, A., Halpern, J. Y., and Koller, D. (1992). From statistics to beliefs. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 602–608, San Jose. AAAI Press.

Bacchus, F. and van Beek, P. (1998). On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 311–318, Madison, Wisconsin. AAAI Press.

Bacchus, F. and van Run, P. (1995). Dynamic variable ordering in CSPs. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pp. 258–275, Cassis, France. Springer-Verlag.

Bachmann, P. G. H. (1894). *Die analytische Zahlentheorie*. B. G. Teubner, Leipzig.

Backus, J. W. (1996). Transcript of question and answer session. In Wexelblat, R. L. (Ed.), *History of Programming Languages*, p. 162. Academic Press, New York.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley Longman, Reading, Massachusetts.

Bajcsy, R. and Lieberman, L. (1976). Texture gradient as a depth cue. *Computer Graphics and Image Processing*, *5*(1), 52–67.

Baker, C. L. (1989). *English Syntax*. MIT Press, Cambridge, Massachusetts.

Baker, J. (1975). The Dragon system—an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *23*, 24–29.

Baker, J. (1979). Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pp. 547–550, Cambridge, Massachusetts. MIT Press.

Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist*, *30*, 441–451. Continued on pages 536–553.

Ballard, B. W. (1983). The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, *21*(3), 327–350.

Baluja, S. (1997). Genetic algorithms and explicit search statistics. In Mozer, M. C., Jordan, M. I., and Petsche, T. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 9, pp. 319–325. MIT Press, Cambridge, Massachusetts.

Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J. D. (1986). Magic sets and other strange ways to implement logic programs. In *Proceedings of the Fifth ACM Symposium on Principles of Database Systems*, pp. 1–16, New York. ACM Press.

Bar-Hillel, Y. (1954). Indexical expressions. *Mind*, *63*, 359–379.

Bar-Hillel, Y. (1960). The present status of automatic translation of languages. In Alt, F. L. (Ed.), *Advances in Computers*, Vol. 1, pp. 91–163. Academic Press, New York.

Bar-Shalom, Y. (Ed.). (1992). *Multitarget-multisensor tracking: Advanced applications*. Artech House, Norwood, Massachusetts.

Bar-Shalom, Y. and Fortmann, T. E. (1988). *Tracking and Data Association*. Academic Press, New York.

Barrett, A. and Weld, D. S. (1994). Task-decomposition via plan parsing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1117–1122, Seattle. AAAI Press.

Bartak, R. (2001). Theory and practice of constraint propagation. In *Proceedings of the Third Workshop on Constraint Programming for Decision and Control (CPDC-01)*, pp. 7–14, Gliwice, Poland.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *73*(1), 81–138.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, *13*, 834–846.

Barto, A. G., Sutton, R. S., and Brouwer, P. S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, *40*(3), 201–211.

Barton, G. E., Berwick, R. C., and Ristad, E. S. (1987). *Computational Complexity and Natural Language*. MIT Press, Cambridge, Massachusetts.

Barwise, J. and Etchemendy, J. (1993). *The Language of First-Order Logic: Including the Macintosh Program Tarski's World 4.0* (Third Revised and Expanded edition). Center for the Study of Language and Information (CSLI), Stanford, California.

Bateman, J. A. (1997). Enabling technology for multilingual natural language generation: The KPML development environment. *Natural Language Engineering*, *3*(1), 15–55.

Bateman, J. A., Kasper, R. T., Moore, J. D., and Whitney, R. A. (1989). A general organization of knowledge for natural language processing: The penman upper model. Tech. rep., Information Sciences Institute, Marina del Rey, CA.

Baum, E., Boneh, D., and Garrett, C. (1995). On genetic algorithms. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory (COLT-92)*, pp. 230–239, Santa Cruz, California. ACM Press.

Baum, E. and Haussler, D. (1989). What size net gives valid generalization?. *Neural Computation*, *1*(1), 151–160.

Baum, E. and Smith, W. D. (1997). A Bayesian approach to relevance in game playing. *Artificial Intelligence*, *97*(1–2), 195–242.

Baum, E. and Wilczek, F. (1988). Supervised learning of probability distributions by neural networks. In Anderson, D. Z. (Ed.), *Neural Information Processing Systems*, pp. 52–61. American Institute of Physics, New York.

Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, *41*.

Baxter, J. and Bartlett, P. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 41–48, Stanford, California. Morgan Kaufmann.

Bayardo, R. J. and Schrag, R. C. (1997). Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 203–208, Providence, Rhode Island. AAAI Press.

Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, *53*, 370–418.

Beal, D. F. (1980). An analysis of minimax. In Clarke, M. R. B. (Ed.), *Advances in Computer Chess 2*, pp. 103–109. Edinburgh University Press, Edinburgh, Scotland.

Beal, D. F. (1990). A generalised quiescence search algorithm. *Artificial Intelligence*, *43*(1), 85–98.

Beckert, B. and Posegga, J. (1995). Leantap: Lean, tableau-based deduction. *Journal of Automated Reasoning*, *15*(3), 339–358.

Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. (1983). On the desirability of acyclic database schemes. *Journal of the Association for Computing Machinery*, *30*(3), 479–513.

Bell, C. and Tate, A. (1985). Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Alvey IKBS SIG Workshop*, Sunningdale, Oxfordshire, UK. Institution of Electrical Engineers.

Bell, J. L. and Machover, M. (1977). *A Course in Mathematical Logic*. Elsevier/North-Holland, Amsterdam, London, New York.

Bellman, R. E. (1978). *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, San Francisco.

Bellman, R. E. and Dreyfus, S. E. (1962). *Applied Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *24*(4), 509–522.

Bender, E. A. (1996). *Mathematical methods in artificial intelligence*. IEEE Computer Society Press, Los Alamitos, California.

Bentham, J. (1823). *Principles of Morals and Legislation*. Oxford University Press, Oxford, UK. Original work published in 1789.

Berger, J. O. (1985). *Statistical Decision Theory and bayesian Analysis*. Springer Verlag, Berlin.

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways, For Your Mathematical Plays*. Academic Press, New York.

Berleur, J. and Brunnstein, K. (2001). *Ethics of Computing: Codes, Spaces for Discussion and Law*. Chapman and Hall, London.

Berliner, H. J. (1977). BKG—a program that plays backgammon. Tech. rep., Computer Science Department, Carnegie-Mellon University, Pittsburgh.

Berliner, H. J. (1979). The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, *12*(1), 23–40.

Berliner, H. J. (1980a). Backgammon computer program beats world champion. *Artificial Intelligence*, *14*, 205–220.

Berliner, H. J. (1980b). Computer backgammon. *Scientific American*, *249*(6), 64–72.

Berliner, H. J. and Ebeling, C. (1989). Pattern knowledge and search: The SUPREM architecture. *Artificial Intelligence*, *38*(2), 161–198.

Bernardo, J. M. and Smith, A. F. M. (1994). *Bayesian Theory*. Wiley, New York.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, *284*(5), 34–43.

Bernoulli, D. (1738). Specimen theoriae novae de mensura sortis. *Proceedings of the St. Petersburg Imperial Academy of Sciences*, *5*, 175–192.

Bernstein, A. and Roberts, M. (1958). Computer vs. chess player. *Scientific American*, *198*(6), 96–105.

Bernstein, A., Roberts, M., Arbuckle, T., and Belsky, M. S. (1958). A chess playing program for the IBM 704. In *Proceedings of the 1958 Western Joint Computer Conference*, pp. 157–159, Los Angeles. American Institute of Electrical Engineers.

Bernstein, P. L. (1996). *Against the Odds: The Remarkable Story of Risk*. Wiley, New York.

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near Shannon limit error control-correcting coding and decoding: Turbo-codes. 1. In *Proc. IEEE International Conference on Communications*, pp. 1064–1070, Geneva, Switzerland. IEEE.

Berry, D. A. and Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London.

Bertele, U. and Brioschi, F. (1972). *Nonserial dynamic programming*. Academic Press, New York.

Bertoli, P., Cimatti, A., and Roveri, M. (2001a). Heuristic search + symbolic model checking = efficient conformant planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 467–472, Seattle. Morgan Kaufmann.

Bertoli, P., Cimatti, A., Roveri, M., and Traverso, P. (2001b). Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 473–478, Seattle. Morgan Kaufmann.

Bertsekas, D. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Upper Saddle River, New Jersey.

Bertsekas, D. and Tsitsiklis, J. N. (1996). *Neurodynamic programming*. Athena Scientific, Belmont, Massachusetts.

Bertsekas, D. and Tsitsiklis, J. N. (2002). *Introduction to Probability*. Athena Scientific, Belmont, Massachusetts.

Bibel, W. (1981). On matrices with connections. *Journal of the Association for Computing Machinery*, *28*(4), 633–645.

Bibel, W. (1993). *Deduction: Automated Logic*. Academic Press, London.

Biggs, N. L., Lloyd, E. K., and Wilson, R. J. (1986). *Graph Theory 1736–1936*. Oxford University Press, Oxford, UK.

Binder, J., Koller, D., Russell, S. J., and Kanazawa, K. (1997a). Adaptive probabilistic networks with hidden variables. *Machine Learning*, *29*, 213–244.

Binder, J., Murphy, K., and Russell, S. J. (1997b). Space-efficient inference in dynamic probabilistic networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1292–1296, Nagoya, Japan. Morgan Kaufmann.

Binford, T. O. (1971). Visual perception by computer. Invited paper presented at the IEEE Systems Science and Cybernetics Conference, Miami.

Binmore, K. (1982). *Essays on Foundations of Game Theory*. Pitman, London.

Birnbaum, L. and Selfridge, M. (1981). Conceptual analysis of natural language. In Schank, R. and Riesbeck, C. (Eds.), *Inside Computer Understanding*. Lawrence Erlbaum, Potomac, Maryland.

Biro, J. I. and Shahan, R. W. (Eds.). (1982). *Mind, Brain and Function: Essays in the Philosophy of Mind*. University of Oklahoma Press, Norman, Oklahoma.

Birtwistle, G., Dahl, O.-J., Myrhaug, B., and Nygaard, K. (1973). *Simula Begin*. Studentliteratur (Lund) and Auerbach, New York.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK.

Bistarelli, S., Montanari, U., and Rossi, F. (1997). Semiring-based constraint satisfaction and optimization. *Journal of the Association for Computing Machinery*, *44*(2), 201–236.

Bitner, J. R. and Reingold, E. M. (1975). Backtrack programming techniques. *Communications of the Association for Computing Machinery*, *18*(11), 651–656.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2001). Latent Dirichlet Allocation. In *Neural Information Processing Systems*, Vol. 14, Cambridge, Massachusetts. MIT Press.

Blinder, A. S. (1983). Issues in the coordination of monetary and fiscal policies. In *Monetary Policy Issues in the 1980s*. Federal Reserve Bank, Kansas City, Missouri.

Block, N. (Ed.). (1980). *Readings in Philosophy of Psychology*, Vol. 1. Harvard University Press, Cambridge, Massachusetts.

Blum, A. L. and Furst, M. (1995). Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1636–1642, Montreal. Morgan Kaufmann.

Blum, A. L. and Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1–2), 281–300.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, *36*(4), 929–965.

Bobrow, D. G. (1967). Natural language input for a computer problem solving system. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 133–215. MIT Press, Cambridge, Massachusetts.

Bobrow, D. G., Kaplan, R., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, a frame driven dialog system. *Artificial Intelligence*, *8*, 155–173.

Bobrow, D. G. and Raphael, B. (1974). New programming languages for artificial intelligence research. *Computing Surveys*, *6*(3), 153–174.

Boden, M. A. (1977). *Artificial Intelligence and Natural Man*. Basic Books, New York.

Boden, M. A. (Ed.). (1990). *The Philosophy of Artificial Intelligence*. Oxford University Press, Oxford, UK.

Bonet, B. and Geffner, H. (1999). Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, pp. 360–372, Durham, UK. Springer-Verlag.

Bonet, B. and Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In Chien, S., Kambhampati, S., and Knoblock, C. A. (Eds.), *International Conference on Artificial Intelligence Planning and Scheduling*, pp. 52–61, Menlo Park, California. AAAI Press.

Boole, G. (1847). *The Mathematical Analysis of Logic: Being an Essay towards a Calculus of Deductive Reasoning*. Macmillan, Barclay, and Macmillan, Cambridge.

Boolos, G. S. and Jeffrey, R. C. (1989). *Computability and Logic* (3rd edition). Cambridge University Press, Cambridge, UK.

Booth, T. L. (1969). Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pp. 74–81, Waterloo, Ontario. IEEE.

Borel, E. (1921). La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, *173*, 1304–1308.

Borenstein, J., Everett, B., and Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Wellesley, MA.

Borenstein, J. and Koren., Y. (1991). The vector field histogram—fast obstacle avoidance for moile robots. *IEEE Transactions on Robotics and Automation*, *7*(3), 278–288.

Borgida, A., Brachman, R. J., McGuinness, D. L., and Alperin Resnick, L. (1989). CLASSIC: A structural data model for objects. *SIGMOD Record*, *18*(2), 58–67.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92)*, Pittsburgh, Pennsylvania. ACM Press.

Boutilier, C. and Brafman, R. I. (2001). Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, *14*, 105–136.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, *121*, 49–107.

Boutilier, C., Reiter, R., and Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 467–472, Seattle. Morgan Kaufmann.

Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pp. 355–362, Austin, Texas. AAAI Press.

Box, G. E. P. (1957). Evolutionary operation: A method of increasing industrial productivity. *Applied Statistics*, *6*, 81–101.

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, *49*(2–3), 233–246.

Boyan, J. A. and Moore, A. W. (1998). Learning evaluation functions for global optimization and Boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin. AAAI Press.

Boyen, X., Friedman, N., and Koller, D. (1999). Discovering the hidden structure of complex dynamic systems. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, Stockholm. Morgan Kaufmann.

Boyer, R. S. and Moore, J. S. (1979). *A Computational Logic*. Academic Press, New York.

Boyer, R. S. and Moore, J. S. (1984). Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly*, *91*(3), 181–189.

Brachman, R. J. (1979). On the epistemological status of semantic networks. In Findler, N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, pp. 3–50. Academic Press, New York.

Brachman, R. J., Fikes, R. E., and Levesque, H. J. (1983). Krypton: A functional approach to knowledge representation. *Computer*, *16*(10), 67–73.

Brachman, R. J. and Levesque, H. J. (Eds.). (1985). *Readings in Knowledge Representation*. Morgan Kaufmann, San Mateo, California.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*, 33–57.

Brafman, R. I. and Tennenholtz, M. (2000). A near optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artificial Intelligence*, *121*, 31–47.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press.

Bransford, J. and Johnson, M. (1973). Consideration of some problems in comprehension. In Chase, W. G. (Ed.), *Visual Information Processing*. Academic Press, New York.

Bratko, I. (1986). *Prolog Programming for Artificial Intelligence* (1st edition). Addison-Wesley, Reading, Massachusetts.

Bratko, I. (2001). *Prolog Programming for Artificial Intelligence* (Third edition). Addison-Wesley, Reading, Massachusetts.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts.

Bratman, M. E. (1992). Planning and the stability of intention. *Minds and Machines*, *2*(1), 1–16.

Breese, J. S. and Heckerman, D. (1996). Decision-theoretic troubleshooting: A framework for repair and experiment. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 124–132, Portland, Oregon. Morgan Kaufmann.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *26*(2), 123–140.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, P. J. (1984). *Classification and Regression Trees*. Wadsworth International Group, Belmont, California.

Brelaz, D. (1979). New methods to color the vertices of a graph. *Communications of the Association for Computing Machinery*, *22*(4), 251–256.

Brent, R. P. (1973). *Algorithms for minimization without derivatives*. Prentice-Hall, Upper Saddle River, New Jersey.

Bresnan, J. (1982). *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts.

Brewka, G., Dix, J., and Konolige, K. (1997). *Nononotonic Reasoning: An Overview*. CSLI Publications, Stanford, California.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman Soulié, F. and Hérault, J. (Eds.), *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, Berlin.

Briggs, R. (1985). Knowledge representation in Sanskrit and artificial intelligence. *AI Magazine*, *6*(1), 32–39.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh World Wide Web Conference*, Brisbane, Australia.

Broadbent, D. E. (1958). *Perception and communication*. Pergamon, Oxford, UK.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*, 14–23.

Brooks, R. A. (1989). Engineering approach to building complete, intelligent beings. *Proceedings of the SPIE—the International Society for Optical Engineering*, *1002*, 618–625.

Brooks, R. A. (1990). Elephants don't play chess. *Autonomous Robots*, *6*, 3–15.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*(1–3), 139–159.

Brooks, R. A. and Lozano-Perez, T. (1985). A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man and Cybernetics*, *15*(2), 224–233.

Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares, M., and Haussler, D. (2000). Knowledge-based analysis of microarray gene expression data using support vector machines. In *Proceedings of the national Academy of Sciences*, Vol. 97, pp. 262–267.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Mercer, R. L., and Roossin, P. (1988). A statistical approach to language translation. In *Proceedings of the 12th International Conference on Computational Linguistics*, pp. 71–76, Budapest. John von Neumann Society for Computing Sciences.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, *19*(2), 263–311.

Brownston, L., Farrell, R., Kant, E., and Martin, N. (1985). *Programming expert systems in OPS5: An introduction to rule-based programming*. Addison-Wesley, Reading, Massachusetts.

Brudno, A. L. (1963). Bounds and valuations for shortening the scanning of variations. *Problems of Cybernetics*, *10*, 225–241.

Bruner, J. S., Goodnow, J. J., and Austin, G. A. (1957). *A Study of Thinking*. Wiley, New York.

Bryant, R. E. (1992). Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, *24*(3), 293–318.

Bryson, A. E. and Ho, Y.-C. (1969). *Applied Optimal Control*. Blaisdell, New York.

Buchanan, B. G. and Mitchell, T. M. (1978). Model-directed learning of production rules. In Waterman, D. A. and Hayes-Roth, F. (Eds.), *Pattern-Directed Inference Systems*, pp. 297–312. Academic Press, New York.

Buchanan, B. G., Mitchell, T. M., Smith, R. G., and Johnson, C. R. (1978). Models of learning systems. In *Encyclopedia of Computer Science and Technology*, Vol. 11. Dekker, New York.

Buchanan, B. G. and Shortliffe, E. H. (Eds.). (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Massachusetts.

Buchanan, B. G., Sutherland, G. L., and Feigenbaum, E. A. (1969). Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 209–254. Edinburgh University Press, Edinburgh, Scotland.

Bundy, A. (1999). A survey of automated deduction. In Wooldridge, M. J. and Veloso, M. (Eds.), *Artificial intelligence today: Recent trends and developments*, pp. 153–174. Springer-Verlag, Berlin.

Bunt, H. C. (1985). The formal representation of (quasi-) continuous concepts. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 2, pp. 37–70. Ablex, Norwood, New Jersey.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, *114*(1-2), 3–55.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, *134*(1–2), 85–99.

Burstall, R. M. (1974). Program proving as hand simulation with a little induction. In *Information Processing '74*, pp. 308–312. Elsevier/North-Holland, Amsterdam, London, New York.

Burstall, R. M. and Darlington, J. (1977). A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery*, *24*(1), 44–67.

Burstein, J., Leacock, C., and Swartz, R. (2001). Automated evaluation of essays and short answers. In *Fifth International Computer Assisted Assessment (CAA) Conference*, Loughborough, U.K. Loughborough University.

Bylander, T. (1992). Complexity results for serial decomposability. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 729–734, San Jose. AAAI Press.

Bylander, T. (1994). The computational complexity of propositional strips planning. *Artificial Intelligence*, *69*, 165–204.

Calvanese, D., Lenzerini, M., and Nardi, D. (1999). Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research*, *11*, 199–240.

Campbell, M. S., Hoane, A. J., and Hsu, F.-H. (2002). Deep Blue. *Artificial Intelligence*, *134*(1–2), 57–83.

Canny, J. and Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *IEEE Symposium on Foundations of Computer Science*, pp. 39–48.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *8*, 679–698.

Canny, J. (1988). *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, Massachusetts.

Carbonell, J. G. (1983). Derivational analogy and its role in problem solving. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, pp. 64–69, Washington, DC. Morgan Kaufmann.

Carbonell, J. G., Knoblock, C. A., and Minton, S. (1989). PRODIGY: An integrated architecture for planning and learning. Technical report CMU-CS-89-189, Computer Science Department, Carnegie-Mellon University, Pittsburgh.

Carbonell, J. R. and Collins, A. M. (1973). Natural semantics in artificial intelligence. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pp. 344–351, Stanford, California. IJCAII.

Carnap, R. (1928). *Der logische Aufbau der Welt*. Weltkreis-verlag, Berlin-Schlachtensee. Translated into English as (Carnap, 1967).

Carnap, R. (1948). On the application of inductive logic. *Philosophy and Phenomenological Research*, *8*, 133–148.

Carnap, R. (1950). *Logical Foundations of Probability*. University of Chicago Press, Chicago.

Carrasco, R. C., Oncina, J., and Calera, J. (1998). *Stochastic Inference of Regular Tree Languages*, Vol. 1433 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1023–1028, Seattle. AAAI Press.

Ceri, S., Gottlob, G., and Tanca, L. (1990). *Logic programming and databases*. Springer-Verlag, Berlin.

Chakrabarti, P. P., Ghose, S., Acharya, A., and de Sarkar, S. C. (1989). Heuristic search in restricted memory. *Artificial Intelligence*, *41*(2), 197–222.

Chan, W. P., Prete, F., and Dickinson, M. H. (1998). Visual input to the efferent control system of a fly's 'gyroscope'. *Science, 289*, 289–292.

Chandra, A. K. and Harel, D. (1980). Computable queries for relational data bases. *Journal of Computer and System Sciences, 21*(2), 156–178.

Chandra, A. K. and Merlin, P. M. (1977). Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pp. 77–90, New York. ACM Press.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence, 32*(3), 333–377.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.

Charniak, E. (1996). Tree-bank grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1031–1036, Portland, Oregon. AAAI Press.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 598–603, Providence, Rhode Island. AAAI Press.

Charniak, E. and Goldman, R. (1992). A Bayesian model of plan recognition. *Artificial Intelligence, 64*(1), 53–79.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts.

Charniak, E., Riesbeck, C., McDermott, D., and Meehan, J. (1987). *Artificial Intelligence Programming* (2nd edition). Lawrence Erlbaum Associates, Potomac, Maryland.

Chatfield, C. (1989). *The Analysis of Time Series: An Introduction* (4th edition). Chapman and Hall, London.

Cheeseman, P. (1985). In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 1002–1009, Los Angeles. Morgan Kaufmann.

Cheeseman, P. (1988). An inquiry into computer understanding. *Computational Intelligence, 4*(1), 58–66.

Cheeseman, P., Kanefsky, B., and Taylor, W. (1991). Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 331–337, Sydney. Morgan Kaufmann.

Cheeseman, P., Self, M., Kelly, J., and Stutz, J. (1988). Bayesian classification. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Vol. 2, pp. 607–611, St. Paul, Minnesota. Morgan Kaufmann.

Cheeseman, P. and Stutz, J. (1996). Bayesian classification (AutoClass): Theory and results. In Fayyad, U., Piatesky-Shapiro, G., Smyth, P., and Uthurusamy, R. (Eds.), *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, Menlo Park, California.

Cheng, J. and Druzdzel, M. J. (2000). AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research, 13*, 155–188.

Cheng, J., Greiner, R., Kelly, J., Bell, D. A., and Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence, 137*, 43–90.

Chierchia, G. and McConnell-Ginet, S. (1990). *Meaning and Grammar*. MIT Press, Cambridge, Massachusetts.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory, 2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague and Paris.

Chomsky, N. (1980). Rules and representations. *The Behavioral and Brain Sciences, 3*, 1–61.

Choset, H. (1996). *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. Ph.D. thesis, California Institute of Technology.

Chung, K. L. (1979). *Elementary Probability Theory with Stochastic Processes* (3rd edition). Springer-Verlag, Berlin.

Church, A. (1936). A note on the Entscheidungsproblem. *Journal of Symbolic Logic, 1*, 40–41 and 101–102.

Church, K. and Patil, R. (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics, 8*(3–4), 139–149.

Church, K. and Gale, W. A. (1991). A comparison of the enhanced Good–Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language, 5*, 19–54.

Churchland, P. M. (1979). *Scientific Realism and the Plasticity of Mind*. Cambridge University Press, Cambridge, UK.

Churchland, P. M. and Churchland, P. S. (1982). Functionalism, qualia, and intentionality. In Biro, J. I. and Shahan, R. W. (Eds.), *Mind, Brain and Function: Essays in the Philosophy of Mind*, pp. 121–145. University of Oklahoma Press, Norman, Oklahoma.

Churchland, P. S. (1986). *Neurophilosophy: Toward a Unified Science of the Mind–Brain*. MIT Press, Cambridge, Massachusetts.

Cimatti, A., Roveri, M., and Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 875–881, Madison, Wisconsin. AAAI Press.

Clark, K. L. (1978). Negation as failure. In Gallaire, H. and Minker, J. (Eds.), *Logic and Data Bases*, pp. 293–322. Plenum, New York.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–283.

Clarke, A. C. (1968a). *2001: A Space Odyssey*. Signet.

Clarke, A. C. (1968b). The world of 2001. Vogue.

Clarke, E. and Grumberg, O. (1987). Research on automatic verification of finite-state concurrent systems. *Annual Review of Computer Science*, *2*, 269–290.

Clarke, E., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press, Cambridge, Massachusetts.

Clarke, M. R. B. (Ed.). (1977). *Advances in Computer Chess 1*. Edinburgh University Press, Edinburgh, Scotland.

Clearwater, S. H. (Ed.). (1996). *Market-Based Control*. World Scientific, Singapore and Teaneck, New Jersey.

Clocksin, W. F. and Mellish, C. S. (1994). *Programming in Prolog* (4th edition). Springer-Verlag, Berlin.

Clowes, M. B. (1971). On seeing things. *Artificial Intelligence*, *2*(1), 79–116.

Cobham, A. (1964). The intrinsic computational difficulty of functions. In Bar-Hillel, Y. (Ed.), *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pp. 24–30, Jerusalem. Elsevier/North-Holland.

Cobley, P. (1997). *Introducing Semiotics*. Totem Books, New York.

Cohen, J. (1988). A view of the origins and development of PROLOG. *Communications of the Association for Computing Machinery*, *31*, 26–36.

Cohen, P. R. (1995). *Empirical methods for artificial intelligence*. MIT Press, Cambridge, Massachusetts.

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, *42*(2–3), 213–261.

Cohen, P. R., Morgan, J., and Pollack, M. E. (1990). *Intentions in Communication*. MIT Press, Cambridge, Massachusetts.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, *3*(3), 177–212.

Cohen, W. W. and Page, C. D. (1995). Learnability in inductive logic programming: Methods and results. *New Generation Computing*, *13*(3–4), 369–409.

Cohn, A. G., Bennett, B., Gooday, J. M., and Gotts, N. (1997). RCC: A calculus for region based qualitative spatial reasoning. *GeoInformatica*, *1*, 275–316.

Collins, M. J. (1996). A new statistical parser based on bigram lexical dependencies. In Joshi, A. K. and Palmer, M. (Eds.), *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pp. 184–191, San Francisco. Morgan Kaufmann Publishers.

Collins, M. J. (1999). *Head-driven Statistical Models for Natural Language Processing*. Ph.D. thesis, University of Pennsylvania.

Collins, M. and Duffy, K. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the ACL*.

Colmerauer, A. (1975). Les grammaires de metamorphose. Tech. rep., Groupe d'Intelligence Artificielle, Université de Marseille-Luminy.

Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P. (1973). Un systéme de communication homme–machine en Français. Rapport, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II.

Condon, J. H. and Thompson, K. (1982). Belle chess hardware. In Clarke, M. R. B. (Ed.), *Advances in Computer Chess 3*, pp. 45–54. Pergamon, Oxford, UK.

Congdon, C. B., Huber, M., Kortenkamp, D., Bidlack, C., Cohen, C., Huffman, S., Koss, F., Raschke, U., and Weymouth, T. (1992). CARMEL versus Flakey: A comparison of two robots. Tech. rep. Papers from the AAAI Robot Competition, RC-92-01, American Association for Artificial Intelligence, Menlo Park, CA.

Connell, J. (1989). *A Colony Architecture for an Artificial Creature*. Ph.D. thesis, Artificial Intelligence Laboratory, MIT, Cambridge, MA. also available as AI Technical Report 1151.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pp. 151–158, New York. ACM Press.

Cook, S. A. and Mitchell, D. (1997). Finding hard instances of the satisfiability problem: A survey. In Du, D., Gu, J., and Pardalos, P. (Eds.), *Satisfiability problems: Theory and applications*. American Mathematical Society, Providence, Rhode Island.

Cooper, G. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, *42*, 393–405.

Cooper, G. and Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Copeland, J. (1993). *Artificial Intelligence: A Philosophical Introduction*. Blackwell, Oxford, UK.

Copernicus (1543). *De Revolutionibus Orbium Coelestium*. Apud Ioh. Petreium, Nuremberg.

Cormen, T. H., Leiserson, C. E., and Rivest, R. (1990). *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts.

Cortes, C. and Vapnik, V. N. (1995). Support vector networks. *Machine Learning*, *20*, 273–297.

Cournot, A. (Ed.). (1838). *Recherches sur les principes mathématiques de la théorie des richesses*. L. Hachette, Paris.

Covington, M. A. (1994). *Natural Language Processing for Prolog Programmers*. Prentice-Hall, Upper Saddle River, New Jersey.

Cowan, J. D. and Sharp, D. H. (1988a). Neural nets. *Quarterly Reviews of Biophysics*, *21*, 365–427.

Cowan, J. D. and Sharp, D. H. (1988b). Neural nets and artificial intelligence. *Daedalus*, *117*, 85–121.

Cox, I. (1993). A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision*, *10*, 53–66.

Cox, I. and Hingorani, S. L. (1994). An efficient implementation and evaluation of Reid's multiple hypothesis tracking algorithm for visual tracking. In *Proceedings of the 12th International Conference on Pattern Recognition*, Vol. 1, pp. 437–442, Jerusalem, Israel. International Association for Pattern Recognition (IAPR).

Cox, I. and Wilfong, G. T. (Eds.). (1990). *Autonomous Robot Vehicles*. Springer Verlag, Berlin.

Cox, R. T. (1946). Probability, frequency, and reasonable expectation. *American Journal of Physics*, *14*(1), 1–13.

Craig, J. (1989). *Introduction to Robotics: Mechanics and Control (2nd Edition)*. Addison-Wesley Publishing, Inc., Reading, MA.

Craik, K. J. (1943). *The Nature of Explanation*. Cambridge University Press, Cambridge, UK.

Crawford, J. M. and Auton, L. D. (1993). Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 21–27, Washington, DC. AAAI Press.

Cristianini, N. and Schölkopf, B. (2002). Support vector machines and kernel methods: The new generation of learning machines. *AI Magazine*, *23*(3), 31–41.

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, UK.

Crockett, L. (1994). *The Turing Test and the Frame Problem: AI's Mistaken Understanding of Intelligence*. Ablex, Norwood, New Jersey.

Cross, S. E. and Walker, E. (1994). Dart: Applying knowledge based planning and scheduling to crisis action planning. In Zweben, M. and Fox, M. S. (Eds.), *Intelligent Scheduling*, pp. 711–729. Morgan Kaufmann, San Mateo, California.

Cruse, D. A. (1986). *Lexical Semantics*. Cambridge University Press.

Culberson, J. and Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(4), 318–334.

Cullingford, R. E. (1981). Integrating knowledge sources for computer "understanding" tasks. *IEEE Transactions on Systems, Man and Cybernetics (SMC)*, *11*.

Cussens, J. and Dzeroski, S. (2000). *Learning Language in Logic*, Vol. 1925 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, Massachusetts.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Controls, Signals, and Systems*, *2*, 303–314.

Daganzo, C. (1979). *Multinomial probit: The theory and its application to demand forecasting*. Academic Press, New York.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, *60*(1), 141–153.

Dahl, O.-J., Myrhaug, B., and Nygaard, K. (1970). (Simula 67) common base language. Tech. rep. N. S-22, Norsk Regnesentral (Norwegian Computing Center), Oslo.

Dale, R., Moisl, H., and Somers, H. (2000). *Handbook of Natural Language Processing*. Marcel Dekker, New York.

Dantzig, G. B. (1949). Programming of interdependent activities: II. mathematical model. *Econometrica*, *17*, 200–211.

Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence*, *126*, 5–41.

Darwiche, A. and Ginsberg, M. L. (1992). A symbolic generalization of probability theory. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 622–627, San Jose. AAAI Press.

Darwin, C. (1859). *On The Origin of Species by Means of Natural Selection*. J. Murray, London.

Darwin, C. (1871). *Descent of Man*. J. Murray.

Dasgupta, P., Chakrabarti, P. P., and DeSarkar, S. C. (1994). Agent searching in a tree and the optimality of iterative deepening. *Artificial Intelligence*, *71*, 195–208.

Davidson, D. (1980). *Essays on Actions and Events*. Oxford University Press, Oxford, UK.

Davies, T. R. (1985). Analogy. Informal note IN-CSLI-85-4, Center for the Study of Language and Information (CSLI), Stanford, California.

Davies, T. R. and Russell, S. J. (1987). A logical approach to reasoning by analogy. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Vol. 1, pp. 264–270, Milan. Morgan Kaufmann.

Davis, E. (1986). *Representing and Acquiring Geographic Knowledge*. Pitman and Morgan Kaufmann, London and San Mateo, California.

Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann, San Mateo, California.

Davis, K. H., Biddulph, R., and Balashek, S. (1952). Automatic recognition of spoken digits. *Journal of the Acoustical Society of America*, *24*(6), 637–642.

Davis, M. (1957). A computer program for Presburger's algorithm. In Robinson, A. (Ed.), *Proving Theorems (as Done by Man, Logician, or Machine)*, pp. 215–233, Cornell University, Ithaca, New York. Communications Research Division, Institute for Defense Analysis. Proceedings of the Summer Institute for Symbolic Logic. Second edition; publication date is 1960.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, *5*, 394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, *7*(3), 201–215.

Davis, R. and Lenat, D. B. (1982). *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, New York.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, *8*(3–4), 341–362.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, Cambridge, Massachusetts.

de Dombal, F. T., Leaper, D. J., Horrocks, J. C., and Staniland, J. R. (1974). Human and computer-aided diagnosis of abdominal pain: Further report with emphasis on performance of clinicians. *British Medical Journal*, *1*, 376–380.

de Dombal, F. T., Staniland, J. R., and Clamp, S. E. (1981). Geographical variation in disease presentation. *Medical Decision Making*, *1*, 59–69.

de Finetti, B. (1937). Le prévision: ses lois logiques, ses sources subjectives. *Ann. Inst. Poincaré*, *7*, 1–68.

de Freitas, J. F. G., Niranjan, M., and Gee, A. H. (2000). Sequential Monte Carlo methods to train neural network models. *Neural Computation*, *12*(4), 933–953.

de Kleer, J. (1975). Qualitative and quantitative knowledge in classical mechanics. Tech. rep. AI-TR-352, MIT Artificial Intelligence Laboratory.

de Kleer, J. (1986a). An assumption-based TMS. *Artificial Intelligence*, *28*(2), 127–162.

de Kleer, J. (1986b). Extending the ATMS. *Artificial Intelligence*, *28*(2), 163–196.

de Kleer, J. (1986c). Problem solving with the ATMS. *Artificial Intelligence*, *28*(2), 197–224.

de Kleer, J. (1989). A comparison of ATMS and CSP techniques. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 1, pp. 290–296, Detroit. Morgan Kaufmann.

de Kleer, J. and Brown, J. S. (1985). A qualitative physics based on confluences. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 4, pp. 109–183. Ablex, Norwood, New Jersey.

de Marcken, C. (1996). *Unsupervised Language Acquisition*. Ph.D. thesis, MIT.

De Morgan, A. (1864). On the syllogism IV and on the logic of relations. *Cambridge Philosophical Transactions*, *x*, 331–358.

De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, New York.

de Saussure, F. (1910 (republished 1993)). *Lectures on General Linguistics.* Pergamon Press, Oxford, UK.

Deacon, T. W. (1997). *The symbolic species: The co-evolution of language and the brain.* W. W. Norton, New York.

Deale, M., Yvanovich, M., Schnitzius, D., Kautz, D., Carpenter, M., Zweben, M., Davis, G., and Daun, B. (1994). The space shuttle ground processing scheduling system. In Zweben, M. and Fox, M. (Eds.), *Intelligent Scheduling*, pp. 423–449. Morgan Kaufmann, San Mateo, California.

Dean, T., Basye, K., Chekaluk, R., and Hyun, S. (1990). Coping with uncertainty in a control system for navigation and exploration. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, pp. 1010–1015, Boston. MIT Press.

Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pp. 49–54, St. Paul, Minnesota. Morgan Kaufmann.

Dean, T., Firby, R. J., and Miller, D. (1990). Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, *6*(1), 381–398.

Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. (1993). Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 574–579, Washington, DC. AAAI Press.

Dean, T. and Kanazawa, K. (1989a). A model for projection and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 985–990, Detroit. Morgan Kaufmann.

Dean, T. and Kanazawa, K. (1989b). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*(3), 142–150.

Dean, T., Kanazawa, K., and Shewchuk, J. (1990). Prediction, observation and estimation in planning and control. In *5th IEEE International Symposium on Intelligent Control*, Vol. 2, pp. 645–650, Los Alamitos, CA. IEEE Computer Society Press.

Dean, T. and Wellman, M. P. (1991). *Planning and Control.* Morgan Kaufmann, San Mateo, California.

Debevec, P., Taylor, C., and Malik, J. (1996). Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics (SIGGRAPH)*, pp. 11–20.

Debreu, G. (1960). Topological methods in cardinal utility theory. In Arrow, K. J., Karlin, S., and Suppes, P. (Eds.), *Mathematical Methods in the Social Sciences, 1959.* Stanford University Press, Stanford, California.

Dechter, R. (1990a). Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, *41*, 273–312.

Dechter, R. (1990b). On the expressiveness of networks with hidden variables. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 379–385, Boston. MIT Press.

Dechter, R. (1992). Constraint networks. In Shapiro, S. (Ed.), *Encyclopedia of Artificial Intelligence* (2nd edition)., pp. 276–285. Wiley and Sons, New York.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*, 41–85.

Dechter, R. and Frost, D. (1999). Backtracking algorothms for constraint satisfaction problems. Tech. rep., Department of Information and Computer Science, University of California, Irvine.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery*, *32*(3), 505–536.

Dechter, R. and Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*(1), 1–38.

Dechter, R. and Pearl, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, *38*(3), 353–366.

DeCoste, D. and Scholkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, *46*(1), 161–190.

Dedekind, R. (1888). *Was sind und was sollen die Zahlen.* Braunschweig, Germany.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, *41*(6), 391–407.

DeGroot, M. H. (1970). *Optimal Statistical Decisions.* McGraw-Hill, New York.

DeGroot, M. H. (1989). *Probability and Statistics* (2nd edition). Addison-Wesley, Reading, Massachusetts.

DeJong, G. (1981). Generalizations based on explanations. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pp. 67–69, Vancouver, British Columbia. Morgan Kaufmann.

DeJong, G. (1982). An overview of the FRUMP system. In Lehnert, W. and Ringle, M. (Eds.), *Strategies for Natural Language Processing*, pp. 149–176. Lawrence Erlbaum, Potomac, Maryland.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*, 145–176.

Dempster, A. P. (1968). A generalization of Bayesian inference. *Journal of the Royal Statistical Society*, *30 (Series B)*, 205–247.

Dempster, A. P., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *39 (Series B)*, 1–38.

Denes, P. (1959). The design and operation of the mechanical speech recognizer at University College London. *Journal of the British Institution of Radio Engineers*, *19*(4), 219–234.

Deng, X. and Papadimitriou, C. H. (1990). Exploring an unknown graph. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, pp. 355–361, St. Louis. IEEE Computer Society Press.

Denis, F. (2001). Learning regular languages from simple positive examples. *Machine Learning*, *44*(1/2), 37–66.

Dennett, D. C. (1971). Intentional systems. *The Journal of Philosophy*, *68*(4), 87–106.

Dennett, D. C. (1978). Why you can't make a computer that feels pain. *Synthese*, *38*(3).

Dennett, D. C. (1984). Cognitive wheels: the frame problem of AI. In Hookway, C. (Ed.), *Minds, Machines, and Evolution: Philosophical Studies*, pp. 129–151. Cambridge University Press, Cambridge, UK.

Deo, N. and Pang, C.-Y. (1984). Shortest path algorithms: Taxonomy and annotation. *Networks*, *14*(2), 275–323.

Descartes, R. (1637). Discourse on method. In Cottingham, J., Stoothoff, R., and Murdoch, D. (Eds.), *The Philosophical Writings of Descartes*, Vol. I. Cambridge University Press, Cambridge, UK.

Descotte, Y. and Latombe, J.-C. (1985). Making compromises among antagonist constraints in a planner. *Artificial Intelligence*, *27*, 183–217.

Devroye, L. (1987). *A course in density estimation*. Birkhauser, Boston.

Devroye, L., Gyorfi, L., and Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. Springer-Verlag, Berlin.

Dickmanns, E. D. and Zapp, A. (1987). Autonomous high speed road vehicle guidance by computer vision. In Isermann, R. (Ed.), *Automatic Control—World Congress, 1987: Selected Papers from the 10th Triennial World Congress of the International Federation of Automatic Control*, pp. 221–226, Munich. Pergamon.

Dietterich, T. (1990). Machine learning. *Annual Review of Computer Science*, *4*, 255–306.

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

DiGioia, A. M., Kanade, T., and Wells, P. (1996). Final report of the second international workshop on robotics and computer assisted medical interventions. *Computer Aided Surgery*, *2*, 69–101.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M. (2001). A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions of Robotics and Automation*, *17*(3), 229–241.

Do, M. B. and Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporal planner. In *Proccedings of the European Conference on Planning*, Toledo, Spain. Springer-Verlag.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero–one loss. *Machine Learning*, *29*, 103–30.

Doran, C., Egedi, D., Hockey, B. A., Srinivas, B., and Zaidel, M. (1994). XTAG system—a wide coverage grammar of English. In Nagao, M. (Ed.), *Proceedings of the 15th COLING*, Kyoto, Japan.

Doran, J. and Michie, D. (1966). Experiments with the graph traverser program. *Proceedings of the Royal Society of London*, *294, Series A*, 235–259.

Dorf, R. C. and Bishop, R. H. (1999). *Modern Control Systems*. Addison-Wesley, Reading, Massachusetts.

Doucet, A. (1997). *Monte Carlo methods for Bayesian estimation of hidden Markov models: Application to radiation signals*. Ph.D. thesis, Université de Paris-Sud, Orsay, France.

Dowling, W. F. and Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulas. *Journal of Logic Programming*, *1*, 267–284.

Dowty, D., Wall, R., and Peters, S. (1991). *Introduction to Montague Semantics*. D. Reidel, Dordrecht, Netherlands.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, *12*(3), 231–272.

Doyle, J. (1983). What is rational psychology? Toward a modern mental philosophy. *AI Magazine*, *4*(3), 50–53.

Doyle, J. and Patil, R. (1991). Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, *48*(3), 261–297.

Drabble, B. (1990). Mission scheduling for spacecraft: Diaries of T-SCHED. In *Expert Planning Systems*, pp. 76–81, Brighton, UK. Institute of Electrical Engineers.

Draper, D., Hanks, S., and Weld, D. S. (1994). Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on AI Planning Systems*, pp. 31–36, Chicago. Morgan Kaufmann.

Dreyfus, H. L. (1972). *What Computers Can't Do: A Critique of Artificial Reason*. Harper and Row, New York.

Dreyfus, H. L. (1992). *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press, Cambridge, Massachusetts.

Dreyfus, H. L. and Dreyfus, S. E. (1986). *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. Blackwell, Oxford, UK.

Dreyfus, S. E. (1969). An appraisal of some shortest-paths algorithms. *Operations Research*, *17*, 395–412.

Du, D., Gu, J., and Pardalos, P. M. (Eds.). (1999). *Optimization methods for logical inference*. American Mathematical Society, Providence, Rhode Island.

Dubois, D. and Prade, H. (1994). A survey of belief revision and updating rules in various uncertainty models. *International Journal of Intelligent Systems*, *9*(1), 61–100.

Duda, R. O., Gaschnig, J., and Hart, P. E. (1979). Model design in the Prospector consultant system for mineral exploration. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*, pp. 153–167. Edinburgh University Press, Edinburgh, Scotland.

Duda, R. O. and Hart, P. E. (1973). *Pattern classification and scene analysis*. Wiley, New York.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley, New York.

Dudek, G. and Jenkin, M. (2000). *Computational Principles of Mobile Robotics*. Cambridge University Press, Cambridge CB2 2RU, UK.

Durfee, E. H. and Lesser, V. R. (1989). Negotiating task decomposition and allocation using partial global planning. In Huhns, M. and Gasser, L. (Eds.), *Distributed AI*, Vol. 2. Morgan Kaufmann, San Mateo, California.

Dyer, M. (1983). *In-Depth Understanding*. MIT Press, Cambridge, Massachusetts.

Dzeroski, S., Muggleton, S. H., and Russell, S. J. (1992). PAC-learnability of determinate logic programs. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT-92)*, pp. 128–135, Pittsburgh, Pennsylvania. ACM Press.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, *13*(2), 94–102.

Ebeling, C. (1987). *All the Right Moves*. MIT Press, Cambridge, Massachusetts.

Eco, U. (1979). *Theory of Semiotics*. Indiana University Press, Bloomington, Indiana.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, *17*, 449–467.

Edwards, P. (Ed.). (1967). *The Encyclopedia of Philosophy*. Macmillan, London.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1998). The KR system dlv: Progress report, comparisons and benchmarks. In Cohn, A., Schubert, L., and Shapiro, S. (Eds.), *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 406–417, Trento, Italy.

Elhadad, M. (1993). FUF: The universal unifier—user manual. Technical report, Ben Gurion University of the Negev, Be'er Sheva, Israel.

Elkan, C. (1993). The paradoxical success of fuzzy logic. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 698–703, Washington, DC. AAAI Press.

Elkan, C. (1997). Boosting and naive Bayesian learning. Tech. rep., Department of Computer Science and Engineering, University of California, San Diego.

Elman, J., Bates, E., Johnson, M., Karmiloff-Smith, A., Parisi, D., and Plunkett, K. (1997). *Rethinking Innateness*. MIT Press, Cambridge, Massachusetts.

Empson, W. (1953). *Seven Types of Ambiguity*. New Directions, New York.

Enderton, H. B. (1972). *A Mathematical Introduction to Logic*. Academic Press, New York.

Erdmann, M. A. and Mason, M. (1988). An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, *4*(4), 369–379.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, R. (1980). The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, *12*(2), 213–253.

Ernst, H. A. (1961). *MH-1, a Computer-Operated Mechanical Hand*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Ernst, M., Millstein, T., and Weld, D. S. (1997). Automatic SAT-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1169–1176, Nagoya, Japan. Morgan Kaufmann.

Erol, K., Hendler, J., and Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1123–1128, Seattle. AAAI Press.

Erol, K., Hendler, J., and Nau, D. S. (1996). Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, *18*(1), 69–93.

Etzioni, O. (1989). Tractable decision-analytic control. In *Proc. of 1st International Conference on Knowledge Representation and Reasoning*, pp. 114–125, Toronto.

Etzioni, O., Hanks, S., Weld, D. S., Draper, D., Lesh, N., and Williamson, M. (1992). An approach to planning with incomplete information. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts.

Etzioni, O. and Weld, D. S. (1994). A softbot-based interface to the Internet. *Communications of the Association for Computing Machinery*, *37*(7), 72–76.

Evans, T. G. (1968). A program for the solution of a class of geometric-analogy intelligence-test questions. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 271–353. MIT Press, Cambridge, Massachusetts.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge*. MIT Press, Cambridge, Massachusetts.

Fahlman, S. E. (1974). A planning system for robot construction tasks. *Artificial Intelligence*, *5*(1), 1–49.

Fahlman, S. E. (1979). *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press, Cambridge, Massachusetts.

Faugeras, O. (1992). What can be seen in three dimensions with an uncalibrated stereo rig?. In Sandini, G. (Ed.), *Proceedings of the European Conference on Computer Vision*, Vol. 588 of *Lecture Notes in Computer Science*, pp. 563–578. Springer-Verlag.

Faugeras, O. (1993). *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts.

Faugeras, O., Luong, Q.-T., and Papadopoulo, T. (2001). *The Geometry of Multiple Images*. MIT Press, Cambridge, Massachusetts.

Fearing, R. S. and Hollerbach, J. M. (1985). Basic solid mechanics for tactile sensing. *International Journal of Robotics Research*, *4*(3), 40–54.

Featherstone, R. (1987). *Robot Dynamics Algorithms*. Kluwer Academic Publishers, Boston, MA.

Feigenbaum, E. A. (1961). The simulation of verbal learning behavior. *Proceedings of the Western Joint Computer Conference*, *19*, 121–131.

Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6*, pp. 165–190. Edinburgh University Press, Edinburgh, Scotland.

Feigenbaum, E. A. and Feldman, J. (Eds.). (1963). *Computers and Thought*. McGraw-Hill, New York.

Feldman, J. and Sproull, R. F. (1977). Decision theory and artificial intelligence II: The hungry monkey. Technical report, Computer Science Department, University of Rochester.

Feldman, J. and Yakimovsky, Y. (1974). Decision theory and artificial intelligence I: Semantics-based region analyzer. *Artificial Intelligence*, *5*(4), 349–371.

Fellbaum, C. (2001). *Wordnet: An Electronic Lexical Database*. MIT Press, Cambridge, Massachusetts.

Feller, W. (1971). *An Introductioon to Probability Theory and its Applications*, Vol. 2. John Wiley.

Ferraris, P. and Giunchiglia, E. (2000). Planning as satisability in nondeterministic domains. In *Proceedings of Seventeenth National Conference on Artificial Intelligence*, pp. 748–753. AAAI Press.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*(4), 251–288.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3–4), 189–208.

Fikes, R. E. and Nilsson, N. J. (1993). STRIPS, a retrospective. *Artificial Intelligence*, *59*(1–2), 227–232.

Findlay, J. N. (1941). Time: A treatment of some puzzles. *Australasian Journal of Psychology and Philosophy*, *19*(3), 216–235.

Finney, D. J. (1947). *Probit analysis: A statistical treatment of the sigmoid response curve*. Cambridge University Press, Cambridge, UK.

Firby, J. (1994). Task networks for controlling continuous processes. In Hammond, K. (Ed.), *Proceedings of the Second International Conference on AI Planning Systems*, pp. 49–54, Menlo Park, CA. AAAI Press.

Firby, R. J. (1996). Modularity issues in reactive planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pp. 78–85, Edinburgh, Scotland. AAAI Press.

Fischer, M. J. and Ladner, R. E. (1977). Propositional modal logic of programs. In *Proceedings of the 9th ACM Symposium on the Theory of Computing*, pp. 286–294, New York. ACM Press.

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London, Series A 222*, 309–368.

Fix, E. and Hodges, J. L. (1951). Discriminatory analysis—nonparametric discrimination: Consistency properties. Tech. rep. 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas.

Fogel, D. B. (2000). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, New Jersey.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. Wiley, New York.

Forbes, J. (2002). *Learning Optimal Control for Autonomous Vehicles*. Ph.D. thesis, University of California, Berkeley.

Forbus, K. D. (1985). The role of qualitative dynamics in naive physics. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 5, pp. 185–226. Ablex, Norwood, New Jersey.

Forbus, K. D. and de Kleer, J. (1993). *Building Problem Solvers*. MIT Press, Cambridge, Massachusetts.

Ford, K. M. and Hayes, P. J. (1995). Turing Test considered harmful. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 972–977, Montreal. Morgan Kaufmann.

Forestier, J.-P. and Varaiya, P. (1978). Multilayer control of large Markov chains. *IEEE Transactions on Automatic Control, 23*(2), 298–304.

Forgy, C. (1981). OPS5 user's manual. Technical report CMU-CS-81-135, Computer Science Department, Carnegie-Mellon University, Pittsburgh.

Forgy, C. (1982). A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence, 19*(1), 17–37.

Forsyth, D. and Zisserman, A. (1991). Reflections on shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 13*(7), 671–679.

Fortescue, M. D. (1984). *West Greenlandic*. Croom Helm, London.

Foster, D. W. (1989). *Elegy by W. W.: A Study in Attribution*. Associated University Presses, Cranbury, New Jersey.

Fourier, J. (1827). Analyse des travaux de l'Académie Royale des Sciences, pendant l'année 1824; partie mathématique. *Histoire de l'Académie Royale des Sciences de France, 7*, xlvii–lv.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Orlando, FL. AAAI.

Fox, M. S. (1990). Constraint-guided scheduling: A short history of research at CMU. *Computers in Industry, 14*(1–3), 79–88.

Fox, M. S., Allen, B., and Strohm, G. (1982). Job shop scheduling: An investigation in constraint-directed reasoning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pp. 155–158, Pittsburgh, Pennsylvania. Morgan Kaufmann.

Fox, M. S. and Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research, 9*, 367–421.

Frakes, W. and Baeza-Yates, R. (Eds.). (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Upper Saddle River, New Jersey.

Francis, S. and Kucera, H. (1967). *Computing Analysis of Present-day American English*. Brown University Press, Providence, Rhode Island.

Franco, J. and Paull, M. (1983). Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics, 5*, 77–87.

Frank, R. H. and Cook, P. J. (1996). *The Winner-Take-All Society*. Penguin, New York.

Frege, G. (1879). *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, Berlin. English translation appears in van Heijenoort (1967).

Freuder, E. C. (1978). Synthesizing constraint expressions. *Communications of the Association for Computing Machinery, 21*(11), 958–966.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, *29*(1), 24–32.

Freuder, E. C. (1985). A sufficient condition for backtrack-bounded search. *Journal of the Association for Computing Machinery*, *32*(4), 755–761.

Freuder, E. C. and Mackworth, A. K. (Eds.). (1994). *Constraint-based reasoning*. MIT Press, Cambridge, Massachusetts.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy. Morgan Kaufmann.

Friedberg, R. M. (1958). A learning machine: Part I. *IBM Journal*, *2*, 2–13.

Friedberg, R. M., Dunham, B., and North, T. (1959). A learning machine: Part II. *IBM Journal of Research and Development*, *3*(3), 282–287.

Friedman, G. J. (1959). Digital simulation of an evolutionary process. *General Systems Yearbook*, *4*, 171–184.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, *28*(2), 337–374.

Friedman, N. (1998). The Bayesian structural EM algorithm. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, Madison, Wisconsin. Morgan Kaufmann.

Friedman, N. and Goldszmidt, M. (1996). Learning Bayesian networks with local structure. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 252–262, Portland, Oregon. Morgan Kaufmann.

Fry, D. B. (1959). Theoretical aspects of mechanical speech recognition. *Journal of the British Institution of Radio Engineers*, *19*(4), 211–218.

Fuchs, J. J., Gasquet, A., Olalainty, B., and Currie, K. W. (1990). PlanERS-1: An expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems*, pp. 70–75, Brighton, UK. Institute of Electrical Engineers.

Fudenberg, D. and Tirole, J. (1991). *Game theory*. MIT Press, Cambridge, Massachusetts.

Fukunaga, A. S., Rabideau, G., Chien, S., and Yan, D. (1997). ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space*, pp. 181–187, Tokyo.

Fung, R. and Chang, K. C. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, pp. 209–220, Windsor, Ontario. Morgan Kaufmann.

Gaifman, H. (1964). Concerning measures in first order calculi. *Israel Journal of Mathematics*, *2*, 1–18.

Gallaire, H. and Minker, J. (Eds.). (1978). *Logic and Databases*. Plenum, New York.

Gallier, J. H. (1986). *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row, New York.

Gallo, G. and Pallottino, S. (1988). Shortest path algorithms. *Annals of Operations Research*, *13*, 3–79.

Gamba, A., Gamberini, L., Palmieri, G., and Sanna, R. (1961). Further experiments with PAPA. *Nuovo Cimento Supplemento*, *20*(2), 221–231.

Garding, J. (1992). Shape from texture for smooth curved surfaces in perspective projection. *Journal of Mathematical Imaging and Vision*, *2*(4), 327–350.

Gardner, M. (1968). *Logic Machines, Diagrams and Boolean Algebra*. Dover, New York.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*. W. H. Freeman, New York.

Gaschnig, J. (1977). A general backtrack algorithm that eliminates most redundant tests. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, p. 457, Cambridge, Massachusetts. IJCAII.

Gaschnig, J. (1979). Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Computer Science Department, Carnegie-Mellon University.

Gasser, R. (1995). *Efficiently harnessing computational resources for exhaustive search*. Ph.D. thesis, ETH Zürich, Switzerland.

Gasser, R. (1998). Solving nine men's morris. In Nowakowski, R. (Ed.), *Games of No Chance*. Cambridge University Press, Cambridge, UK.

Gat, E. (1998). Three-layered architectures. In Kortenkamp, D., Bonasso, R. P., and Murphy, R. (Eds.), *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, pp. 195–210. MIT Press.

Gauss, K. F. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Sumtibus F. Perthes et I. H. Besser, Hamburg.

Gauss, K. F. (1829). Beiträge zur theorie der algebraischen gleichungen. Collected in *Werke, Vol. 3*, pages 71–102. K. Gesellschaft Wissenschaft, Göttingen, Germany, 1876.

Gawande, A. (2002). *Complications: A Surgeon's Notes on an Imperfect Science*. Metropolitan Books, New York.

Ge, N., Hale, J., and Charniak, E. (1998). A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pp. 161–171, Montreal. COLING-ACL.

Geiger, D., Verma, T., and Pearl, J. (1990). Identifying independence in Bayesian networks. *Networks*, *20*(5), 507–534.

Gelb, A. (1974). *Applied Optimal Estimation*. MIT Press, Cambridge, Massachusetts.

Gelernter, H. (1959). Realization of a geometry-theorem proving machine. In *Proceedings of an International Conference on Information Processing*, pp. 273–282, Paris. UNESCO House.

Gelfond, M. and Lifschitz, V. (1988). Compiling circumscriptive theories into logic programs. In Reinfrank, M., de Kleer, J., Ginsberg, M. L., and Sandewall, E. (Eds.), *Non-Monotonic Reasoning: 2nd International Workshop Proceedings*, pp. 74–99, Grassau, Germany. Springer-Verlag.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. (1995). *Bayesian Data Analysis*. Chapman & Hall, London.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images.. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *6*(6), 721–741.

Genesereth, M. R. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence*, *24*(1–3), 411–436.

Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.

Genesereth, M. R. and Nourbakhsh, I. (1993). Time-saving tips for problem solving with incomplete information. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 724–730, Washington, DC. AAAI Press.

Genesereth, M. R. and Smith, D. E. (1981). Meta-level architecture. Memo HPP-81-6, Computer Science Department, Stanford University, Stanford, California.

Gentner, D. (1983). Structure mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155–170.

Gentzen, G. (1934). Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, *39*, 176–210, 405–431.

Georgeff, M. P. and Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pp. 677–682, Seattle. Morgan Kaufmann.

Gerevini, A. and Schubert, L. K. (1996). Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research*, *5*, 95–137.

Gerevini, A. and Serina, I. (2002). LPG: A planner based on planning graphs with action costs. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, pp. 281–290, Menlo Park, California. AAAI Press.

Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast decoding and optimal decoding for machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pp. 228–235, Toulouse, France.

Gershwin, G. (1937). Let's call the whole thing off. song.

Ghahramani, Z. and Jordan, M. I. (1997). Factorial hidden Markov models. *Machine Learning*, *29*, 245–274.

Ghallab, M., Howe, A., Knoblock, C. A., and McDermott, D. (1998). PDDL—the planning domain definition language. Tech. rep. DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut.

Ghallab, M. and Laruelle, H. (1994). Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pp. 61–67, Chicago. AAAI Press.

Giacomo, G. D., Lespérance, Y., and Levesque, H. J. (2000). ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, *121*, 109–169.

Gibson, J. J. (1950). *The Perception of the Visual World*. Houghton Mifflin, Boston.

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston.

Gibson, J. J., Olum, P., and Rosenblatt, F. (1955). Parallax and perspective during aircraft landings. *American Journal of Psychology*, *68*, 372–385.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (Eds.). (1996). *Markov chain Monte Carlo in practice*. Chapman and Hall, London.

Gilks, W. R., Thomas, A., and Spiegelhalter, D. J. (1994). A language and program for complex Bayesian modelling. *The Statistician*, *43*, 169–178.

Gilmore, P. C. (1960). A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, *4*, 28–35.

Ginsberg, M. L. (1989). Universal planning: An (almost) universally bad idea. *AI Magazine*, *10*(4), 40–44.

Ginsberg, M. L. (1993). *Essentials of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.

Ginsberg, M. L. (1999). GIB: Steps toward an expert-level bridge-playing program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 584–589, Stockholm. Morgan Kaufmann.

Ginsberg, M. L., Frank, M., Halpin, M. P., and Torrance, M. C. (1990). Search lessons learned from crossword puzzles. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 1, pp. 210–215, Boston. MIT Press.

Gittins, J. C. (1989). *Multi-Armed Bandit Allocation Indices*. Wiley, New York.

Glanc, A. (1978). On the etymology of the word "robot". *SIGART Newsletter*, *67*, 12.

Glover, F. (1989). Tabu search: 1. *ORSA Journal on Computing*, *1*(3), 190–206.

Glover, F. and Laguna, M. (Eds.). (1997). *Tabu search*. Kluwer, Dordrecht, Netherlands.

Gödel, K. (1930). *Über die Vollständigkeit des Logikkalküls*. Ph.D. thesis, University of Vienna.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, *38*, 173–198.

Goebel, J., Volk, K., Walker, H., and Gerbault, F. (1989). Automatic classification of spectra from the infrared astronomical satellite (IRAS). *Astronomy and Astrophysics*, *222*, L5–L8.

Gold, B. and Morgan, N. (2000). *Speech and Audio Signal Processing*. Wiley, New York.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, *10*, 447–474.

Golden, K. (1998). Leap before you look: Information gathering in the PUCCINI planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pp. 70–77, Pittsburgh, Pennsylvania. AAAI Press.

Goldman, N. (1975). Conceptual generation. In Schank, R. (Ed.), *Conceptual Information Processing*, chap. 6. North-Holland, Amsterdam.

Goldman, R. and Boddy, M. (1996). Expressive planning and explicit knowledge. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pp. 110–117, Edinburgh, Scotland. AAAI Press.

Gomes, C., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 431–437, Madison, Wisconsin. AAAI Press.

Good, I. J. (1950). Contribution to the discussion of Eliot Slater's "Statistics for the chess computer and the factor of mobility". In *Symposium on Information Theory*, p. 199, London. Ministry of Supply.

Good, I. J. (1961). A causal calculus. *British Journal of the Philosophy of Science*, *11*, 305–318.

Good, I. J. (1965). Speculations concerning the first ultraintelligent machine. In Alt, F. L. and Rubinoff, M. (Eds.), *Advances in Computers*, Vol. 6, pp. 31–88. Academic Press, New York.

Goodman, D. and Keene, R. (1997). *Man versus Machine: Kasparov versus Deep Blue*. H3 Publications, Cambridge, Massachusetts.

Goodman, N. (1954). *Fact, Fiction and Forecast*. University of London Press, London.

Goodman, N. (1977). *The Structure of Appearance* (3rd edition). D. Reidel, Dordrecht, Netherlands.

Gordon, M. J., Milner, A. J., and Wadsworth, C. P. (1979). *Edinburgh LCF*. Springer-Verlag, Berlin.

Gordon, N. J. (1994). *Bayesian methods for tracking*. Ph.D. thesis, Imperial College, University of London.

Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, *140*(2), 107–113.

Gorry, G. A. (1968). Strategies for computer-aided diagnosis. *Mathematical Biosciences*, *2*(3–4), 293–318.

Gorry, G. A., Kassirer, J. P., Essig, A., and Schwartz, W. B. (1973). Decision analysis as the basis for computer-aided management of acute renal failure. *American Journal of Medicine*, *55*, 473–484.

Gottlob, G., Leone, N., and Scarcello, F. (1999a). A comparison of structural CSP decomposition methods. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 394–399, Stockholm. Morgan Kaufmann.

Gottlob, G., Leone, N., and Scarcello, F. (1999b). Hypertree decompositions and tractable queries. In *Proceedings of the 18th ACM International Symposium on Principles of Database Systems*, pp. 21–32, Philadelphia. Association for Computing Machinery.

Graham, S. L., Harrison, M. A., and Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, *2*(3), 415–462.

Grassmann, H. (1861). *Lehrbuch der Arithmetik*. Th. Chr. Fr. Enslin, Berlin.

Grayson, C. J. (1960). Decisions under uncertainty: Drilling decisions by oil and gas operators. Tech. rep., Division of Research, Harvard Business School, Boston.

Green, B., Wolf, A., Chomsky, C., and Laugherty, K. (1961). BASEBALL: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference*, pp. 219–224.

Green, C. (1969a). Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, pp. 219–239, Washington, DC. IJCAII.

Green, C. (1969b). Theorem-proving by resolution as a basis for question-answering systems. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 183–205. Edinburgh University Press, Edinburgh, Scotland.

Green, C. and Raphael, B. (1968). The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 23rd ACM National Conference*, Washington, DC. ACM Press.

Greenblatt, R. D., Eastlake, D. E., and Crocker, S. D. (1967). The Greenblatt chess program. In *Proceedings of the Fall Joint Computer Conference*, pp. 801–810. American Federation of Information Processing Societies (AFIPS).

Greiner, R. (1989). Towards a formal analysis of EBL. In *Proceedings of the Sixth International Machine Learning Workshop*, pp. 450–453, Ithaca, NY. Morgan Kaufmann.

Grice, H. P. (1957). Meaning. *Philosophical Review*, *66*, 377–388.

Grosz, B. J., Joshi, A. K., and Weinstein, S. (1995). Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, *21*(2), 203–225.

Grosz, B. J. and Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, *12*(3), 175–204.

Grosz, b. J., Sparck Jones, K., and Webber, B. L. (Eds.). (1986). *Readings in Natural Language Processing*. Morgan Kaufmann, San Mateo, California.

Grove, W. and Meehl, P. (1996). Comparative efficiency of informal (subjective, impressionistic) and formal (mechanical, algorithmic) prediction procedures: The clinical statistical controversy. *Psychology, Public Policy, and Law*, *2*, 293–323.

Gu, J. (1989). *Parallel Algorithms and Architectures for Very Fast AI Search*. Ph.D. thesis, University of Utah.

Guard, J., Oglesby, F., Bennett, J., and Settle, L. (1969). Semi-automated mathematics. *Journal of the Association for Computing Machinery*, *16*, 49–62.

Guibas, L. J., Knuth, D. E., and Sharir, M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, *7*, 381–413. See also *17th Int. Coll. on Automata, Languages and Programming*, 1990, pp. 414–431.

Haas, A. (1986). A syntactic theory of belief and action. *Artificial Intelligence*, *28*(3), 245–292.

Hacking, I. (1975). *The Emergence of Probability*. Cambridge University Press, Cambridge, UK.

Hald, A. (1990). *A History of Probability and Statistics and Their Applications before 1750*. Wiley, New York.

Halpern, J. Y. (1990). An analysis of first-order logics of probability. *Artificial Intelligence*, *46*(3), 311–350.

Hamming, R. W. (1991). *The Art of Probability for Scientists and Engineers*. Addison-Wesley, Reading, Massachusetts.

Hammond, K. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, New York.

Hamscher, W., Console, L., and Kleer, J. D. (1992). *Readings in Model-based Diagnosis*. Morgan Kaufmann, San Mateo, California.

Handschin, J. E. and Mayne, D. Q. (1969). Monte Carlo techniques to estimate the conditional expectation in multi-stage nonlinear filtering. *International Journal of Control*, *9*(5), 547–559.

Hansen, E. (1998). Solving POMDPs by searching in policy space. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, pp. 211–219, Madison, Wisconsin. Morgan Kaufmann.

Hansen, E. and Zilberstein, S. (2001). LAO*: a heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*(1–2), 35–62.

Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing*, *44*(4), 279–303.

Hanski, I. and Cambefort, Y. (Eds.). (1991). *Dung Beetle Ecology*. Princeton University Press, Princeton, New Jersey.

Hansson, O. and Mayer, A. (1989). Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario. Morgan Kaufmann.

Hansson, O., Mayer, A., and Yung, M. (1992). Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences*, *63*(3), 207–227.

Haralick, R. M. and Elliot, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, *14*(3), 263–313.

Hardin, G. (1968). The tragedy of the commons. *Science*, *162*, 1243–1248.

Harel, D. (1984). Dynamic logic. In Gabbay, D. and Guenthner, F. (Eds.), *Handbook of Philosophical Logic*, Vol. 2, pp. 497–604. D. Reidel, Dordrecht, Netherlands.

Harman, G. H. (1983). *Change in View: Principles of Reasoning*. MIT Press, Cambridge, Massachusetts.

Harsanyi, J. (1967). Games with incomplete information played by Bayesian players. *Management Science*, *14*, 159–182.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *SSC-4(2)*, 100–107.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to "A formal basis for the heuristic determination of minimum cost paths". *SIGART Newsletter*, *37*, 28–29.

Hart, T. P. and Edwards, D. J. (1961). The tree prune (TP) algorithm. Artificial intelligence project memo 30, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Hartley, R. and Zisserman, A. (2000). *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, UK.

Haslum, P. and Geffner, H. (2001). Heuristic planning with time and resources. In *Proceedings of the IJCAI-01 Workshop on Planning with Resources*, Seattle.

Hastie, T. and Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification and regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 8, pp. 409–15. MIT Press, Cambridge, Massachusetts.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer-Verlag, Berlin.

Haugeland, J. (Ed.). (1981). *Mind Design*. MIT Press, Cambridge, Massachusetts.

Haugeland, J. (Ed.). (1985). *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, Massachusetts.

Haussler, D. (1989). Learning conjunctive concepts in structural domains. *Machine Learning*, *4*(1), 7–40.

Havelund, K., Lowry, M., Park, S., Pecheur, C., Penix, J., Visser, W., and White, J. L. (2000). Formal analysis of the remote agent before and after flight. In *Proceedings of the 5th NASA Langley Formal Methods Workshop*, Williamsburg, VA.

Hayes, P. J. (1978). The naive physics manifesto. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.

Hayes, P. J. (1979). The logic of frames. In Metzing, D. (Ed.), *Frame Conceptions and Text Understanding*, pp. 46–61. de Gruyter, Berlin.

Hayes, P. J. (1985a). Naive physics I: Ontology for liquids. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 3, pp. 71–107. Ablex, Norwood, New Jersey.

Hayes, P. J. (1985b). The second naive physics manifesto. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, chap. 1, pp. 1–36. Ablex, Norwood, New Jersey.

Hebb, D. O. (1949). *The Organization of Behavior*. Wiley, New York.

Heckerman, D. (1986). Probabilistic interpretation for MYCIN's certainty factors. In Kanal, L. N. and Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence*, pp. 167–196. Elsevier/North-Holland, Amsterdam, London, New York.

Heckerman, D. (1991). *Probabilistic Similarity Networks*. MIT Press, Cambridge, Massachusetts.

Heckerman, D. (1998). A tutorial on learning with Bayesian networks. In Jordan, M. I. (Ed.), *Learning in graphical models*. Kluwer, Dordrecht, Netherlands.

Heckerman, D., Geiger, D., and Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical report MSR-TR-94-09, Microsoft Research, Redmond, Washington.

Heim, I. and Kratzer, A. (1998). *Semantics in a Generative Grammar*. Blackwell, Oxford, UK.

Heinz, E. A. (2000). *Scalable search in computer chess*. Vieweg, Braunschweig, Germany.

Held, M. and Karp, R. M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research*, *18*, 1138–1162.

Helmert, M. (2001). On the complexity of planning in transportation domains. In Cesta, A. and Barrajo, D. (Eds.), *Sixth European Conference on Planning (ECP-01)*, Toledo, Spain. Springer-Verlag.

Hendrix, G. G. (1975). Expanding the utility of semantic networks through partitioning. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pp. 115–121, Tbilisi, Georgia. IJCAII.

Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F. and Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence 2*, pp. 149–163. Elsevier/North-Holland, Amsterdam, London, New York.

Henzinger, T. A. and Sastry, S. (Eds.). (1998). *Hybrid systems: Computation and control*. Springer-Verlag, Berlin.

Herbrand, J. (1930). *Recherches sur la Théorie de la Démonstration*. Ph.D. thesis, University of Paris.

Hewitt, C. (1969). PLANNER: a language for proving theorems in robots. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, pp. 295–301, Washington, DC. IJCAII.

Hierholzer, C. (1873). Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, *6*, 30–32.

Hilgard, E. R. and Bower, G. H. (1975). *Theories of Learning* (4th edition). Prentice-Hall, Upper Saddle River, New Jersey.

Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press, Ithaca, New York.

Hinton, G. E. and Anderson, J. A. (1981). *Parallel Models of Associative Memory*. Lawrence Erlbaum Associates, Potomac, Maryland.

Hinton, G. E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, *1*(3), 495–502.

Hinton, G. E. and Sejnowski, T. (1983). Optimal perceptual inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 448–453, Washington, DC. IEEE Computer Society Press.

Hinton, G. E. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, chap. 7, pp. 282–317. MIT Press, Cambridge, Massachusetts.

Hirsh, H. (1987). Explanation-based generalization in a logic programming environment. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan. Morgan Kaufmann.

Hirst, G. (1981). *Anaphora in Natural Language Understanding: A Survey*, Vol. 119 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin.

Hirst, G. (1987). *Semantic Interpretation against Ambiguity*. Cambridge University Press, Cambridge, UK.

Hobbs, J. R. (1978). Resolving pronoun references. *Lingua*, *44*, 339–352.

Hobbs, J. R. (1990). *Literature and Cognition*. CSLI Press, Stanford, California.

Hobbs, J. R., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M. E., and Tyson, M. (1997). FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In Roche, E. and Schabes, Y. (Eds.), *Finite-State Devices for Natural Language Processing*, pp. 383–406. MIT Press, Cambridge, Massachusetts.

Hobbs, J. R. and Moore, R. C. (Eds.). (1985). *Formal Theories of the Commonsense World*. Ablex, Norwood, New Jersey.

Hobbs, J. R., Stickel, M. E., Appelt, D., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence*, *63*(1–2), 69–142.

Hoffmann, J. (2000). A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems*, pp. 216–227, Charlotte, North Carolina. Springer-Verlag.

Hogan, N. (1985). Impedance control: An approach to manipulation. parts i, ii, and iii. *Transactions ASME Journal of Dynamics, Systems, Measurement, and Control*, *107*(3), 1–24.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan.

Holland, J. H. (1995). *Hidden order: How adaptation builds complexity*. Addison-Wesley, Reading, Massachusetts.

Holldobler, S. and Schneeberger, J. (1990). A new deductive approach to planning. *New Generation Computing*, *8*(3), 225–244.

Holzmann, G. J. (1997). The Spin model checker. *ISSS Transactions on Software Engineering*, *23*(5), 279–295.

Hood, A. (1824). Case 4th—28 July 1824 (Mr. Hood's cases of injuries of the brain). *The Phrenological Journal and Miscellany*, *2*, 82–94.

Hopfield, J. J. (1982). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America*, *79*, 2554–2558.

Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, *16*, 14–21.

Horn, B. K. P. (1970). Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical report 232, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.

Horn, B. K. P. (1986). *Robot Vision*. MIT Press, Cambridge, Massachusetts.

Horn, B. K. P. and Brooks, M. J. (1989). *Shape from Shading*. MIT Press, Cambridge, Massachusetts.

Horning, J. J. (1969). *A study of grammatical inference*. Ph.D. thesis, Stanford University.

Horowitz, E. and Sahni, S. (1978). *Fundamentals of computer algorithms*. Computer Science Press, Rockville, Maryland.

Horswill, I. (2000). Functional programming of behavior-based systems. *Autonomous Robots*, *9*, 83–93.

Horvitz, E. J. (1987). Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proceedings of the Second Annual NASA Research Forum*, pp. 26–43, Moffett Field, California. NASA Ames Research Center.

Horvitz, E. J. (1989). Rational metareasoning and compilation for optimizing decisions under bounded resources. In *Proceedings of Computational Intelligence 89*, Milan. Association for Computing Machinery.

Horvitz, E. J. and Barry, M. (1995). Display of information for time-critical decision making. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, pp. 296–305, Montreal, Canada. Morgan Kaufmann.

Horvitz, E. J., Breese, J. S., Heckerman, D., and Hovel, D. (1998). The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, pp. 256–265, Madison, Wisconsin. Morgan Kaufmann.

Horvitz, E. J., Breese, J. S., and Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, *2*, 247–302.

Horvitz, E. J. and Breese, J. S. (1996). Ideal partition of resources for metareasoning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1229–1234, Portland, Oregon. AAAI Press.

Horvitz, E. J. and Heckerman, D. (1986). The inconsistent use of measures of certainty in artificial intelligence research. In Kanal, L. N. and Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence*, pp. 137–151. Elsevier/North-Holland, Amsterdam, London, New York.

Horvitz, E. J., Heckerman, D., and Langlotz, C. P. (1986). A framework for comparing alternative formalisms for plausible reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 210–214, Philadelphia. Morgan Kaufmann.

Hovy, E. (1988). *Generating Natural Language under Pragmatic Constraints*. Lawrence Erlbaum, Potomac, Maryland.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts.

Howard, R. A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, *SSC-2*, 22–26.

Howard, R. A. (1977). Risk preference. In Howard, R. A. and Matheson, J. E. (Eds.), *Readings in Decision Analysis*, pp. 429–465. Decision Analysis Group, SRI International, Menlo Park, California.

Howard, R. A. (1989). Microrisks for medical decision analysis. *International Journal of Technology Assessment in Health Care*, *5*, 357–370.

Howard, R. A. and Matheson, J. E. (1984). Influence diagrams. In Howard, R. A. and Matheson, J. E. (Eds.), *Readings on the Principles and Applications of Decision Analysis*, pp. 721–762. Strategic Decisions Group, Menlo Park, California.

Hsu, F.-H. (1999). IBM's Deep Blue chess grandmaster chips. *IEEE Micro*, *19*(2), 70–80.

Hsu, F.-H., Anantharaman, T. S., Campbell, M. S., and Nowatzyk, A. (1990). A grandmaster chess machine. *Scientific American*, *263*(4), 44–50.

Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S. J., and Weber, J. (1994). Automatic symbolic traffic scene analysis using belief networks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 966–972, Seattle. AAAI Press.

Huang, X. D., Acero, A., and Hon, H. (2001). *Spoken Language Processing*. Prentice Hall, Upper Saddle River, New Jersey.

Hubel, D. H. (1988). *Eye, Brain, and Vision*. W. H. Freeman, New York.

Huddleston, R. D. and Pullum, G. K. (2002). *The Cambridge Grammar of the English Language*. Cambridge University Press, Cambridge, UK.

Huffman, D. A. (1971). Impossible objects as nonsense sentences. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6*, pp. 295–324. Edinburgh University Press, Edinburgh, Scotland.

Hughes, B. D. (1995). *Random Walks and Random Environments, Vol. 1: Random Walks*. Oxford University Press, Oxford, UK.

Huhns, M. N. and Singh, M. P. (Eds.). (1998). *Readings in agents*. Morgan Kaufmann, San Mateo, California.

Hume, D. (1739). *A Treatise of Human Nature* (2nd edition). republished by Oxford University Press, 1978, Oxford, UK.

Hunsberger, L. and Grosz, B. J. (2000). A combinatorial auction for collaborative planning. In *International Conference on Multi-Agent Systems (ICMAS-2000)*.

Hunt, E. B., Marin, J., and Stone, P. T. (1966). *Experiments in Induction*. Academic Press, New York.

Hunter, L. and States, D. J. (1992). Bayesian classification of protein structure. *IEEE Expert*, *7*(4), 67–75.

Hurwicz, L. (1973). The design of mechanisms for resource allocation. *American Economic Review Papers and Proceedings*, *63*(1), 1–30.

Hutchins, W. J. and Somers, H. (1992). *An Introduction to Machine Translation*. Academic Press, New York.

Huttenlocher, D. P. and Ullman, S. (1990). Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, *5*(2), 195–212.

Huygens, C. (1657). Ratiociniis in ludo aleae. In van Schooten, F. (Ed.), *Exercitionum Mathematicorum*. Elsevirii, Amsterdam.

Hwa, R. (1998). An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *Proceedings of COLING-ACL '98*, pp. 557–563, Montreal. International Committee on Computational Linguistics and Association for Computational Linguistics.

Hwang, C. H. and Schubert, L. K. (1993). EL: A formal, yet natural, comprehensive knowledge representation. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 676–682, Washington, DC. AAAI Press.

Indyk, P. (2000). Dimensionality reduction techniques for proximity problems. In *Proceedings of the Eleventh Annual ACM–SIAM Symposium on Discrete Algorithms*, pp. 371–378, San Francisco. Association for Computing Machinery.

Ingerman, P. Z. (1967). Panini–Backus form suggested. *Communications of the Association for Computing Machinery*, *10*(3), 137.

Inoue, K. (2001). Inverse entailment for full clausal theories. In *LICS-2001 Workshop on Logic and Learning*, Boston. IEEE.

Intille, S. and Bobick, A. (1999). A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 518–525, Orlando, Florida. AAAI Press.

Isard, M. and Blake, A. (1996). Contour tracking by stochastic propagation of conditional density. In *Proceedings of Fourth European Conference on Computer Vision*, pp. 343–356, Cambridge, UK. Springer-Verlag.

Jaakkola, T. and Jordan, M. I. (1996). Computing upper and lower bounds on likelihoods in intractable networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 340–348. Morgan Kaufmann, Portland, Oregon.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In Tesauro, G., Touretzky, D., and Leen, T. (Eds.), *Advances in Neural Information Processing Systems 7*, pp. 345–352, Cambridge, Massachusetts. MIT Press.

Jaffar, J. and Lassez, J.-L. (1987). Constraint logic programming. In *Proceedings of the Fourteenth ACM Conference on Principles of Programming Languages*, pp. 111–119, Munich. Association for Computing Machinery.

Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C. (1992a). The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems*, *14*(3), 339–395.

Jaffar, J., Stuckey, P. J., Michaylov, S., and Yap, R. H. C. (1992b). An abstract machine for CLP(R). *SIGPLAN Notices*, *27*(7), 128–139.

Jáskowski, S. (1934). On the rules of suppositions in formal logic. *Studia Logica*, *1*.

Jefferson, G. (1949). The mind of mechanical man: The Lister Oration delivered at the Royal College of Surgeons in England. *British Medical Journal*, *1*(25), 1105–1121.

Jeffrey, R. C. (1983). *The Logic of Decision* (2nd edition). University of Chicago Press, Chicago.

Jeffreys, H. (1948). *Theory of Probability*. Oxford, Oxford, UK.

Jelinek, F. (1969). Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, *64*, 532–556.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, *64*(4), 532–556.

Jelinek, F. (1997). *Statistical methods for speech recognition*. MIT Press, Cambridge, Massachusetts.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, pp. 381–397, Amsterdam, London, New York. North Holland.

Jennings, H. S. (1906). *Behavior of the lower organisms*. Columbia University Press, New York.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer-Verlag, Berlin.

Jespersen, O. (1965). *Essentials of English Grammar*. University of Alabama Press, Tuscaloosa, Alabama.

Jevons, W. S. (1874). *The Principles of Science*. Routledge/Thoemmes Press, London.

Jimenez, P. and Torras, C. (2000). An efficient algorithm for searching implicit AND/OR graphs with cycles. *Artificial Intelligence*, *124*(1), 1–30.

Joachims, T. (2001). A statistical learning model of text classification with support vector machines. In *Proceedings of the 24th Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 128–136, New Orleans. Association for Computing Machinery.

Johnson, W. W. and Story, W. E. (1879). Notes on the "15" puzzle. *American Journal of Mathematics*, *2*, 397–404.

Johnson-Laird, P. N. (1988). *The Computer and the Mind: An Introduction to Cognitive Science*. Harvard University Press, Cambridge, Massachusetts.

Johnston, M. D. and Adorf, H.-M. (1992). Scheduling with neural networks: The case of the Hubble space telescope. *Computers & Operations Research*, *19*(3–4), 209–240.

Jones, N. D., Gomard, C. K., and Sestoft, P. (1993). *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Upper Saddle River, New Jersey.

Jones, R., Laird, J. E., and Nielsen, P. E. (1998). Automated intelligent pilots for combat flight simulation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 1047–54, Madison, Wisconsin. AAAI Press.

Jonsson, A., Morris, P., Muscettola, N., Rajan, K., and Smith, B. (2000). Planning in interplanetary space: Theory and practice. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pp. 177–186, Breckenridge, Colorado. AAAI Press.

Jordan, M. I. (1995). Why the logistic function? a tutorial discussion on probabilities and neural networks. Computational cognitive science technical report 9503, Massachusetts Institute of Technology.

Jordan, M. I. (2003). *An Introduction to Graphical Models*. In press.

Jordan, M. I., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1998). An introduction to variational methods for graphical models. In Jordan, M. I. (Ed.), *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands.

Jordan, M. I., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, *37*(2–3), 183–233.

Joshi, A. K. (1985). Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions. In Dowty, D., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing*. Cambridge University Press, Cambridge, UK.

Joshi, A. K., Webber, B. L., and Sag, I. (1981). *Elements of Discourse Understanding*. Cambridge University Press, Cambridge, UK.

Joslin, D. and Pollack, M. E. (1994). Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, p. 1506, Seattle. AAAI Press.

Jouannaud, J.-P. and Kirchner, C. (1991). Solving equations in abstract algebras: A rule-based survey of unification. In Lassez, J.-L. and Plotkin, G. (Eds.), *Computational Logic*, pp. 257–321. MIT Press, Cambridge, Massachusetts.

Judd, J. S. (1990). *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, Massachusetts.

Juels, A. and Wattenberg, M. (1996). Stochastic hill-climbing as a baseline method for evaluating genetic algorithms. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 8, pp. 430–6. MIT Press, Cambridge, Massachusetts.

Julesz, B. (1971). *Foundations of Cyclopean Perception*. University of Chicago Press, Chicago.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, Upper Saddle River, New Jersey.

Kadane, J. B. and Larkey, P. D. (1982). Subjective probability and the theory of games. *Management Science*, *28*(2), 113–120.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and actiong in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. *Robotics and Autonomous Systems*, *6*(1–2), 35–48.

Kager, R. (1999). *Optimality Theory*. Cambridge University Press, Cambridge, UK.

Kahneman, D., Slovic, P., and Tversky, A. (Eds.). (1982). *Judgment under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge, UK.

Kaindl, H. and Khorsand, A. (1994). Memory-bounded bidirectional search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1359–1364, Seattle. AAAI Press.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, *82*, 35–46.

Kambhampati, S. (1994). Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, *10*, 213–244.

Kambhampati, S., Mali, A. D., and Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 882–888, Madison, Wisconsin. AAAI Press.

Kanal, L. N. and Kumar, V. (1988). *Search in Artificial Intelligence*. Springer-Verlag, Berlin.

Kanal, L. N. and Lemmer, J. F. (Eds.). (1986). *Uncertainty in Artificial Intelligence*. Elsevier/North-Holland, Amsterdam, London, New York.

Kanazawa, K., Koller, D., and Russell, S. J. (1995). Stochastic simulation algorithms for dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, pp. 346–351, Montreal, Canada. Morgan Kaufmann.

Kaplan, D. and Montague, R. (1960). A paradox regained. *Notre Dame Journal of Formal Logic*, *1*(3), 79–90.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, *4*, 373–395.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W. (Eds.), *Complexity of Computer Computations*, pp. 85–103. Plenum, New York.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.

Kasparov, G. (1997). IBM owes me a rematch. *Time*, *149*(21), 66–67.

Kasper, R. T. (1988). Systemic grammar and functional unification grammar. In Benson, J. and Greaves, W. (Eds.), *Systemic Functional Approaches to Discourse*. Ablex, Norwood, New Jersey.

Kaufmann, M., Manolios, P., and Moore, J. S. (2000). *Computer-Aided Reasoning: An Approach*. Kluwer, Dordrecht, Netherlands.

Kautz, H., McAllester, D. A., and Selman, B. (1996). Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 374–384, Cambridge, Massachusetts. Morgan Kaufmann.

Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pp. 359–363, Vienna. Wiley.

Kautz, H. and Selman, B. (1998). BLACKBOX: A new approach to the application of theorem proving to problem solving. Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search.

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580.

Kay, M., Gawron, J. M., and Norvig, P. (1994). *Verbmobil: A Translation System for Face-To-Face Dialog*. CSLI Press, Stanford, California.

Kaye, R. (2000). Minesweeper is NP-complete!. *Mathematical Intelligencer*, *5*(22), 9–15.

Kearns, M. (1990). *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts.

Kearns, M., Mansour, Y., and Ng, A. Y. (2000). Approximate planning in large POMDPs via reusable trajectories. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*. MIT Press, Cambridge, Massachusetts.

Kearns, M. and Singh, S. P. (1998). Near-optimal reinforcement learning in polynomial time. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 260–268, Madison, Wisconsin. Morgan Kaufmann.

Kearns, M. and Vazirani, U. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts.

Keeney, R. L. (1974). Multiplicative utility functions. *Operations Research*, *22*, 22–34.

Keeney, R. L. and Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, New York.

Kehler, A. (1997). Probabilistic coreference in information extraction. In Cardie, C. and Weischedel, R. (Eds.), *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pp. 163–173. Association for Computational Linguistics, Somerset, New Jersey.

Kemp, M. (Ed.). (1989). *Leonardo on Painting: An Anthology of Writings*. Yale University Press, New Haven, Connecticut.

Kern, C. and Greenstreet, M. R. (1999). Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems*, *4*(2), 123–193.

Keynes, J. M. (1921). *A Treatise on Probability*. Macmillan, London.

Khatib, O. (1986). Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, *5*(1), 90–98.

Kietz, J.-U. and Dzeroski, S. (1994). Inductive logic programming and learnability. *SIGART Bulletin*, *5*(1), 22–32.

Kim, J. H. (1983). *CONVINCE: A Conversational Inference Consolidation Engine*. Ph.D. thesis, Department of Computer Science, University of California at Los Angeles.

Kim, J. H. and Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pp. 190–193, Karlsruhe, Germany. Morgan Kaufmann.

Kim, J. H. and Pearl, J. (1987). CONVINCE: A conversational inference consolidation engine. *IEEE Transactions on Systems, Man, and Cybernetics*, *17*(2), 120–132.

King, R. D., Muggleton, S. H., Lewis, R. A., and Sternberg, M. J. E. (1992). Drug design by machine learning: The use of inductive logic programming to model the structure activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences of the United States of America*, *89*(23), 11322–11326.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Kirkpatrick, S. and Selman, B. (1994). Critical behavior in the satisfiability of random Boolean expressions. *Science*, *264*(5163), 1297–1301.

Kirousis, L. M. and Papadimitriou, C. H. (1988). The complexity of recognizing polyhedral scenes. *Journal of Computer and System Sciences*, *37*(1), 14–38.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). RoboCup: The robot world cup initiative. In Johnson, W. L. and Hayes-Roth, B. (Eds.), *Proceedings of the First International Conference on Autonomous Agents*, pp. 340–347, New York. ACM Press.

Kjaerulff, U. (1992). A computational scheme for reasoning in dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference*, pp. 121–129, Stanford, California. Morgan Kaufmann.

Klein, D. and Manning, C. D. (2001). Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn treebank. In *Proceedings of the 39th Annual Meeting of the ACL*.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, *46*(5), 604–632.

Knight, K. (1999). A statistical mt tutorial workbook. prepared in connection with the Johns Hopkins University summer workshop.

Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, pp. 923–928, Boston. MIT Press.

Knuth, D. E. (1968). Semantics for context-free languages. *Mathematical Systems Theory*, *2*(2), 127–145.

Knuth, D. E. (1973). *The Art of Computer Programming* (second edition)., Vol. 2: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts.

Knuth, D. E. (1975). An analysis of alpha–beta pruning. *Artificial Intelligence*, *6*(4), 293–326.

Knuth, D. E. and Bendix, P. B. (1970). Simple word problems in universal algebras. In Leech, J. (Ed.), *Computational Problems in Abstract Algebra*, pp. 263–267. Pergamon, Oxford, UK.

Koditschek, D. (1987). Exact robot navigation by means of potential functions: some topological considerations. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 1–6, Raleigh, North Carolina. IEEE Computer Society Press.

Koehler, J., Nebel, B., Hoffman, J., and Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In *Proceedings of the Fourth European Conference on Planning*, pp. 273–285, Toulouse, France. Springer-Verlag.

Koenderink, J. J. (1990). *Solid Shape*. MIT Press, Cambridge, Massachusetts.

Koenderink, J. J. and van Doorn, A. J. (1975). Invariant properties of the motion parallax field due to the movement of rigid bodies relative to an observer. *Optica Acta*, *22*(9), 773–791.

Koenderink, J. J. and van Doorn, A. J. (1991). Affine structure from motion. *Journal of the Optical Society of America A*, *8*, 377–385.

Koenig, S. (1991). Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Master's report, Computer Science Division, University of California, Berkeley.

Koenig, S. (2000). Exploring unknown environments with real-time search or reinforcement learning. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*. MIT Press, Cambridge, Massachusetts.

Koenig, S. and Simmons, R. (1998). Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *aips98*. AAAI Press, Menlo Park, California.

Kohn, W. (1991). Declarative control architecture. *Communications of the Association for Computing Machinery*, *34*(8), 65–79.

Koller, D., Meggido, N., and von Stengel, B. (1996). Efficient computation of equilibria for extensive two-person games. *Games and Economic Behaviour*, *14*(2), 247–259.

Koller, D. and Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *Artificial Intelligence*, *94*(1–2), 167–215.

Koller, D. and Pfeffer, A. (1998). Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 580–587, Madison, Wisconsin. AAAI Press.

Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 170–178. Morgan Kaufmann.

Kolmogorov, A. N. (1941). Interpolation und extrapolation von stationaren zufalligen folgen. *Bulletin of the Academy of Sciences of the USSR, Ser. Math. 5*, 3–14.

Kolmogorov, A. N. (1950). *Foundations of the Theory of Probability*. Chelsea, New York.

Kolmogorov, A. N. (1963). On tables of random numbers. *Sankhya, the Indian Journal of Statistics, Series A 25*.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problems in Information Transmission*, *1*(1), 1–7.

Kolodner, J. (1983). Reconstructive memory: A computer model. *Cognitive Science*, *7*, 281–328.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, California.

Kondrak, G. and van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, *89*, 365–387.

Konolige, K. (1997). COLBERT: A language for reactive control in Saphira. In *KI-97: Advances in Artificial Intelligence*, LNAI, pp. 31–52. Springer verlag.

Konolige, K. (1982). A first order formalization of knowledge and action for a multi-agent planning system. In Hayes, J. E., Michie, D., and Pao, Y.-H. (Eds.), *Machine Intelligence 10*. Ellis Horwood, Chichester, England.

Koopmans, T. C. (1972). Representation of preference orderings over time. In McGuire, C. B. and Radner, R. (Eds.), *Decision and Organization*. Elsevier/North-Holland, Amsterdam, London, New York.

Korf, R. E. (1985a). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, *27(1)*, 97–109.

Korf, R. E. (1985b). Iterative-deepening A*: An optimal admissible tree search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 1034–1036, Los Angeles. Morgan Kaufmann.

Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence*, *33(1)*, 65–88.

Korf, R. E. (1988). Optimal path finding algorithms. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial Intelligence*, chap. 7, pp. 223–267. Springer-Verlag, Berlin.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence*, *42*(3), 189–212.

Korf, R. E. (1991). Best-first search with limited memory. UCLA Computer Science Annual.

Korf, R. E. (1993). Linear-space best-first search. *Artificial Intelligence*, *62*(1), 41–78.

Korf, R. E. (1995). Space-efficient search algorithms. *ACM Computing Surveys*, *27*(3), 337–339.

Korf, R. E. and Chickering, D. M. (1996). Best-first minimax search. *Artificial Intelligence*, *84*(1–2), 299–337.

Korf, R. E. and Felner, A. (2002). Disjoint pattern database heuristics. *Artificial Intelligence*, *134*(1–2), 9–22.

Korf, R. E. and Zhang, W. (2000). Divide-and-conquer frontier search applied to optimal sequence alignment. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pp. 910–916, Cambridge, Massachusetts. MIT Press.

Kortenkamp, D., Bonasso, R. P., and Murphy, R. (Eds.). (1998). *AI-based Mobile Robots: Case studies of successful robot systems*, Cambridge, MA. MIT Press.

Kotok, A. (1962). A chess playing program for the IBM 7090. Ai project memo 41, MIT Computation Center, Cambridge, Massachusetts.

Koutsoupias, E. and Papadimitriou, C. H. (1992). On the greedy algorithm for satisfiability. *Information Processing Letters*, *43*(1), 53–55.

Kowalski, R. (1974). Predicate logic as a programming language. In *Proceedings of the IFIP-74 Congress*, pp. 569–574. Elsevier/North-Holland.

Kowalski, R. (1979a). Algorithm = logic + control. *Communications of the Association for Computing Machinery*, *22*, 424–436.

Kowalski, R. (1979b). *Logic for Problem Solving*. Elsevier/North-Holland, Amsterdam, London, New York.

Kowalski, R. (1988). The early years of logic programming. *Communications of the Association for Computing Machinery*, *31*, 38–43.

Kowalski, R. and Kuehner, D. (1971). Linear resolution with selection function. *Artificial Intelligence*, *2*(3–4), 227–260.

Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, *4*(1), 67–95.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts.

Koza, J. R. (1994). *Genetic Programming II: Automatic discovery of reusable programs*. MIT Press, Cambridge, Massachusetts.

Koza, J. R., Bennett, F. H., Andre, D., and Keane, M. A. (1999). *Genetic Programming III: Darwinian invention and problem solving*. Morgan Kaufmann, San Mateo, California.

Kraus, S., Ephrati, E., and Lehmann, D. (1991). Negotiation in a non-cooperative environment. *Journal of Experimental and Theoretical Artificial Intelligence*, *3*(4), 255–281.

Kripke, S. A. (1963). Semantical considerations on modal logic. *Acta Philosophica Fennica*, *16*, 83–94.

Krovetz, R. (1993). Viewing morphology as an inference process. In *Proceedings of the Sixteenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 191–202, New York. ACM Press.

Kruppa, E. (1913). Zur Ermittlung eines Objecktes aus zwei Perspektiven mit innerer Orientierung. *Sitz.-Ber. Akad. Wiss., Wien, Math. Naturw., Kl. Abt. IIa*, *122*, 1939–1948.

Kuhn, H. W. (1953). Extensive games and the problem of information. In Kuhn, H. W. and Tucker, A. W. (Eds.), *Contributions to the Theory of Games II*. Princeton University Press, Princeton, New Jersey.

Kuipers, B. J. and Levitt, T. S. (1988). Navigation and mapping in large-scale space. *AI Magazine*, *9*(2), 25–43.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Kumar, P. R. and Varaiya, P. (1986). *Stochastic systems: Estimation, identification, and adaptive control*. Prentice-Hall, Upper Saddle River, New Jersey.

Kumar, V. (1992). Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, *13*(1), 32–44.

Kumar, V. and Kanal, L. N. (1983). A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *Artificial Intelligence*, *21*, 179–198.

Kumar, V. and Kanal, L. N. (1988). The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial Intelligence*, chap. 1, pp. 1–27. Springer-Verlag, Berlin.

Kumar, V., Nau, D. S., and Kanal, L. N. (1988). A general branch-and-bound formulation for AND/OR graph and game tree search. In Kanal, L. N. and Kumar, V. (Eds.), *Search in Artificial Intelligence*, chap. 3, pp. 91–130. Springer-Verlag, Berlin.

Kuper, G. M. and Vardi, M. Y. (1993). On the complexity of queries in the logical data model. *Theoretical Computer Science*, *116*(1), 33–57.

Kurzweil, R. (1990). *The Age of Intelligent Machines*. MIT Press, Cambridge, Massachusetts.

Kurzweil, R. (2000). *The Age of Spiritual Machines*. Penguin.

Kyburg, H. E. (1977). Randomness and the right reference class. *The Journal of Philosophy*, *74*(9), 501–521.

Kyburg, H. E. (1983). The reference class. *Philosophy of Science*, *50*, 374–397.

La Mettrie, J. O. (1748). *L'homme machine*. E. Luzac, Leyde, France.

La Mura, P. and Shoham, Y. (1999). Expected utility networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, pp. 366–373, Stockholm. Morgan Kaufmann.

Ladkin, P. (1986a). Primitives and units for time specification. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 354–359, Philadelphia. Morgan Kaufmann.

Ladkin, P. (1986b). Time representation: a taxonomy of interval relations. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 360–366, Philadelphia. Morgan Kaufmann.

Lafferty, J. and Zhai, C. (2001). Probabilistic relevance models based on document and query generation. In *Proceedings of the Workshop on Language Modeling and Information retrieval*.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, *33*(1), 1–64.

Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. University of Chicago Press, Chicago.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, Chicago.

Lamarck, J. B. (1809). *Philosophie zoologique*. Chez Dentu et L'Auteur, Paris.

Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*. MIT Press, Cambridge, Massachusetts.

Langton, C. (Ed.). (1995). *Artificial life*. MIT Press, Cambridge, Massachusetts.

Laplace, P. (1816). *Essai philosophique sur les probabilités* (3rd edition). Courcier Imprimeur, Paris.

Lappin, S. and Leass, H. J. (1994). An algorithm for pronominal anaphora resolution. *Computational Linguistics*, *20*(4), 535–561.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer, Speech and Language*, *4*, 35–56.

Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, *13*, 129–170.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer, Dordrecht, Netherlands.

Lauritzen, S. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, *19*, 191–201.

Lauritzen, S. (1996). *Graphical models*. Oxford University Press, Oxford, UK.

Lauritzen, S., Dawid, A., Larsen, B., and Leimer, H. (1990). Independence properties of directed Markov fields. *Networks*, *20*(5), 491–505.

Lauritzen, S. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, *B 50*(2), 157–224.

Lauritzen, S. and Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, *17*, 31–57.

Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, England.

Lawler, E. L. (1985). *The traveling salesman problem: A guided tour of combinatorial optimization*. Wiley, New York.

Lawler, E. L., Lenstra, J. K., Kan, A., and Shmoys, D. B. (1992). *The Travelling Salesman Problem*. Wiley Interscience.

Lawler, E. L., Lenstra, J. K., Kan, A., and Shmoys, D. B. (1993). Sequencing and scheduling: algorithms and complexity. In Graves, S. C., Zipkin, P. H., and Kan, A. H. G. R. (Eds.), *Logistics of Production and Inventory: Handbooks in Operations Research and Management Science, Volume 4*, pp. 445–522. North-Holland, Amsterdam.

Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research*, *14(4)*, 699–719.

Lazanas, A. and Latombe, J.-C. (1992). Landmark-based robot navigation. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 816–822, San Jose. AAAI Press.

Le Cun, Y., Jackel, L., Boser, B., and Denker, J. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, *27*(11), 41–46.

LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., Simard, P., and Vapnik, V. N. (1995). Comparison of learning algorithms for handwritten digit recognition. In Fogelman, F. and Gallinari, P. (Eds.), *International Conference on Artificial Neural Networks*, pp. 53–60, Berlin. Springer-Verlag.

Leech, G., Rayson, P., and Wilson, A. (2001). *Word Frequencies in Written and Spoken English: Based on the British National Corpus*. Longman, New York.

Lefkovitz, D. (1960). A strategic pattern recognition program for the game Go. Technical note 60-243, Wright Air Development Division, University of Pennsylvania, Moore School of Electrical Engineering.

Lenat, D. B. (1983). EURISKO: A program that learns new heuristics and domain concepts: The nature of heuristics, III: Program design and results. *Artificial Intelligence*, *21*(1–2), 61–98.

Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, *38*(11).

Lenat, D. B. and Brown, J. S. (1984). Why AM and EURISKO appear to work. *Artificial Intelligence*, *23*(3), 269–294.

Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, Massachusetts.

Leonard, H. S. and Goodman, N. (1940). The calculus of individuals and its uses. *Journal of Symbolic Logic*, *5*(2), 45–55.

Leonard, J. J. and Durrant-Whyte, H. (1992). *Directed sonar sensing for mobile robot navigation*. Kluwer, Dordrecht, Netherlands.

Leśniewski, S. (1916). Podstawy ogólnej teorii mnogości. Moscow.

Lettvin, J. Y., Maturana, H. R., McCulloch, W. S., and Pitts, W. H. (1959). What the frog's eye tells the frog's brain. *Proceedings of the IRE*, *47*(11), 1940–1951.

Letz, R., Schumann, J., Bayerl, S., and Bibel, W. (1992). SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, *8*(2), 183–212.

Levesque, H. J. and Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, *3*(2), 78–93.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997a). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, *31*, 59–84.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997b). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, *31*, 59–84.

Levitt, G. M. (2000). *The Turk, Chess Automaton*. McFarland and Company.

Levy, D. N. L. (Ed.). (1988a). *Computer Chess Compendium*. Springer-Verlag, Berlin.

Levy, D. N. L. (Ed.). (1988b). *Computer Games*. Springer-Verlag, Berlin.

Lewis, D. D. (1998). Naive Bayes at forty: The independence assumption in information retrieval. In *Machine Learning: ECML-98. 10th European Conference on Machine Learning. Proceedings*, pp. 4–15, Chemnitz, Germany. Springer-Verlag.

Lewis, D. K. (1966). An argument for the identity theory. *The Journal of Philosophy*, *63*(1), 17–25.

Lewis, D. K. (1972). General semantics. In Davidson, D. and Harman, G. (Eds.), *Semantics of Natural Language*, pp. 169–218. D. Reidel, Dordrecht, Netherlands.

Lewis, D. K. (1980). Mad pain and Martian pain. In Block, N. (Ed.), *Readings in Philosophy of Psychology*, Vol. 1, pp. 216–222. Harvard University Press, Cambridge, Massachusetts.

Li, C. M. and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 366–371, Nagoya, Japan. Morgan Kaufmann.

Li, M. and Vitanyi, P. M. B. (1993). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin.

Lifschitz, V. (1986). On the semantics of STRIPS. In Georgeff, M. P. and Lansky, A. L. (Eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pp. 1–9, Timberline, Oregon. Morgan Kaufmann.

Lifschitz, V. (2001). Answer set programming and plan generation. *Artificial Intelligence*, *138*(1–2), 39–54.

Lighthill, J. (1973). Artificial intelligence: A general survey. In Lighthill, J., Sutherland, N. S., Needham, R. M., Longuet-Higgins, H. C., and Michie, D. (Eds.), *Artificial Intelligence: A Paper Symposium*. Science Research Council of Great Britain, London.

Lin, F. and Reiter, R. (1997). How to progress a database. *Artificial Intelligence*, *92*(1–2), 131–167.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal, 44(10)*, 2245–2269.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling-salesman problem. *Operations Research, 21*(2), 498–516.

Linden, T. A. (1991). Representing software designs as partially developed plans. In Lowry, M. R. and McCartney, R. D. (Eds.), *Automating Software Design*, pp. 603–625. MIT Press, Cambridge, Massachusetts.

Lindsay, R. K. (1963). Inferential memory as the basis of machines which understand natural language. In Feigenbaum, E. A. and Feldman, J. (Eds.), *Computers and Thought*, pp. 217–236. McGraw-Hill, New York.

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. McGraw-Hill, New York.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pp. 157–163, New Brunswick, NJ. Morgan Kaufmann.

Littman, M. L., Keim, G. A., and Shazeer, N. M. (1999). Solving crosswords with PROVERB. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 914–915, Orlando, Florida. AAAI Press.

Liu, J. S. and Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association, 93*, 1022–1031.

Lloyd, J. W. (1987). *Foundations of Logic Programming*. Springer-Verlag, Berlin.

Locke, J. (1690). *An Essay Concerning Human Understanding*. William Tegg.

Locke, W. N. and Booth, A. D. (1955). *Machine Translation of Languages: Fourteen Essays*. MIT Press, Cambridge, Massachusetts.

Lodge, D. (1984). *Small World*. Penguin Books, New York.

Lohn, J. D., Kraus, W. F., and Colombano, S. P. (2001). Evolutionary optimization of yagi-uda antennas. In *Proceedings of the Fourth International Conference on Evolvable Systems*, pp. 236–243.

Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature, 293*, 133–135.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research, 28*(1–4), 47–66.

Loveland, D. (1968). Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery, 15*(2), 236–251.

Loveland, D. (1970). A linear format for resolution. In *Proceedings of the IRIA Symposium on Automatic Demonstration*, pp. 147–162, Berlin. Springer-Verlag.

Loveland, D. (1984). Automated theorem-proving: A quarter-century review. *Contemporary Mathematics, 29*, 1–45.

Lowe, D. G. (1987). Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence, 31*, 355–395.

Löwenheim, L. (1915). Über möglichkeiten im Relativkalkül. *Mathematische Annalen, 76*, 447–470.

Lowerre, B. T. (1976). *The* HARPY *Speech Recognition System*. Ph.D. thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Lowerre, B. T. and Reddy, R. (1980). The HARPY speech recognition system. In Lea, W. A. (Ed.), *Trends in Speech Recognition*, chap. 15. Prentice-Hall, Upper Saddle River, New Jersey.

Lowry, M. R. and McCartney, R. D. (1991). *Automating Software Design*. MIT Press, Cambridge, Massachusetts.

Loyd, S. (1959). *Mathematical Puzzles of Sam Loyd: Selected and Edited by Martin Gardner*. Dover, New York.

Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. *IEEE Transactions on Computers, C-32*(2), 108–120.

Lozano-Perez, T., Mason, M., and Taylor, R. (1984). Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research, 3*(1), 3–24.

Luby, M., Sinclair, A., and Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters, 47*, 173–180.

Luby, M. and Vigoda, E. (1999). Fast convergence of the glauber dynamics for sampling independent sets. *Random Structures and Algorithms, 15*(3-4), 229–241.

Lucas, J. R. (1961). Minds, machines, and Gödel. *Philosophy, 36*.

Lucas, J. R. (1976). This Gödel is killing me: A rejoinder. *Philosophia, 6*(1), 145–148.

Lucas, P. (1996). Knowledge acquisition for decision-theoretic expert systems. *AISB Quarterly, 94*, 23–33.

Luce, D. R. and Raiffa, H. (1957). *Games and Decisions*. Wiley, New York.

Luger, G. F. (Ed.). (1995). *Computation and intelligence: Collected readings*. AAAI Press, Menlo Park, California.

MacKay, D. J. C. (1992). A practical Bayesian framework for back-propagation networks. *Neural Computation*, *4*(3), 448–472.

Mackworth, A. K. (1973). Interpreting pictures of polyhedral scenes. *Artificial Intelligence*, *4*, 121–137.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, *8*(1), 99–118.

Mackworth, A. K. (1992). Constraint satisfaction. In Shapiro, S. (Ed.), *Encyclopedia of Artificial Intelligence* (second edition)., Vol. 1, pp. 285–293. Wiley, New York.

Mahanti, A. and Daniels, C. J. (1993). A SIMD approach to parallel heuristic search. *Artificial Intelligence*, *60*(2), 243–282.

Majercik, S. M. and Littman, M. L. (1999). Planning under uncertainty via stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pp. 549–556.

Malik, J. (1987). Interpreting line drawings of curved objects. *International Journal of Computer Vision*, *1*(1), 73–103.

Malik, J. and Rosenholtz, R. (1994). Recovering surface curvature and orientation from texture distortion: A least squares algorithm and sensitivity analysis. In Eklundh, J.-O. (Ed.), *Proceedings of the Third European Conf. on Computer Vision*, pp. 353–364, Stockholm. Springer-Verlag.

Malik, J. and Rosenholtz, R. (1997). Computing local surface orientation and shape from texture for curved surfaces. *International Journal of Computer Vision*, *23*(2), 149–168.

Mann, W. C. and Thompson, S. A. (1983). Relational propositions in discourse. Tech. rep. RR-83-115, Information Sciences Institute.

Mann, W. C. and Thompson, S. A. (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text*, *8*(3), 243–281.

Manna, Z. and Waldinger, R. (1971). Toward automatic program synthesis. *Communications of the Association for Computing Machinery*, *14*(3), 151–165.

Manna, Z. and Waldinger, R. (1985). *The Logical Basis for Computer Programming: Volume 1: Deductive Reasoning*. Addison-Wesley, Reading, Massachusetts.

Manna, Z. and Waldinger, R. (1986). Special relations in automated deduction. *Journal of the Association for Computing Machinery*, *33*(1), 1–59.

Manna, Z. and Waldinger, R. (1992). Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering*, *18*(8), 674–704.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Marbach, P. and Tsitsiklis, J. N. (1998). Simulation-based optimization of Markov reward processes. Technical report LIDS-P-2411, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Markov, A. A. (1913). An example of statistical investigation in the text of "Eugene Onegin" illustrating coupling of "tests" in chains. *Proceedings of the Academy of Sciences of St. Petersburg*, *7*.

Maron, M. E. (1961). Automatic indexing: An experimental inquiry. *Journal of the Association for Computing Machinery*, *8*(3), 404–417.

Maron, M. E. and Kuhns, J.-L. (1960). On relevance, probabilistic indexing and information retrieval. *Communications of the ACM*, *7*, 219–244.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, New York.

Marriott, K. and Stuckey, P. J. (1998). *Programming with Constraints: An Introduction*. MIT Press, Cambridge, Massachusetts.

Marsland, A. T. and Schaeffer, J. (Eds.). (1990). *Computers, Chess, and Cognition*. Springer-Verlag, Berlin.

Martelli, A. and Montanari, U. (1976). Unification in linear time and space: A structured presentation. Internal report B 76-16, Istituto di Elaborazione della Informazione, Pisa, Italy.

Martelli, A. and Montanari, U. (1978). Optimizing decision trees through heuristically guided search. *Communications of the Association for Computing Machinery*, *21*, 1025–1039.

Marthi, B., Pasula, H., Russell, S. J., and Peres, Y. (2002). Decayed MCMC filtering. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Conference*, pp. 319–326, Edmonton, Alberta. Morgan Kaufmann.

Martin, J. H. (1990). *A Computational Model of Metaphor Interpretation*. Academic Press, New York.

Martin, P. and Shmoys, D. B. (1996). A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of the 5th International IPCO Conference*, pp. 389–403. Springer-Verlag.

Maslov, S. Y. (1964). An inverse method for establishing deducibility in classical predicate calculus. *Doklady Akademii nauk SSSR*, *159*, 17–20.

Maslov, S. Y. (1967). An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus. *Doklady Akademii nauk SSSR*, *172*, 22–25.

Mason, M. (1993). Kicking the sensing habit. *AI Magazine*, *14*(1), 58–59.

Mason, M. (2001). *Mechanics of Robotic Manipulation*. MIT Press.

Mason, M. and Salisbury, J. (1985). *Robot hands and the mechanics of manipulation*. MIT Press.

Mataric, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, *4*(1), 73–83.

Mates, B. (1953). *Stoic Logic*. University of California Press, Berkeley and Los Angeles.

Maxwell, J. and Kaplan, R. (1993). The interface between phrasal and functional constraints. *Computational Linguistics*, *19*(4), 571–590.

Maxwell, J. and Kaplan, R. (1995). A method for disjunctive constraint satisfaction. In Dalrymple, M., Kaplan, R., Maxwell, J., and Zaenen, A. (Eds.), *Formal Issues in Lexical-Functional Grammar*, No. 47 in CSLI Lecture Note Series, chap. 14, pp. 381–481. CSLI Publications.

McAllester, D. A. (1980). An outlook on truth maintenance. Ai memo 551, MIT AI Laboratory, Cambridge, Massachusetts.

McAllester, D. A. (1988). Conspiracy numbers for min-max search. *Artificial Intelligence*, *35*(3), 287–310.

McAllester, D. A. (1989). *Ontic: A Knowledge Representation System for Mathematics*. MIT Press, Cambridge, Massachusetts.

McAllester, D. A. (1998). What is the most pressing issue facing ai and the aaai today?. Candidate statement, election for Councilor of the American Association for Artificial Intelligence.

McAllester, D. A. and Givan, R. (1992). Natural language syntax and first-order inference. *Artificial Intelligence*, *56*(1), 1–20.

McAllester, D. A. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Vol. 2, pp. 634–639, Anaheim, California. AAAI Press.

McCarthy, J. (1958). Programs with common sense. In *Proceedings of the Symposium on Mechanisation of Thought Processes*, Vol. 1, pp. 77–84, London. Her Majesty's Stationery Office.

McCarthy, J. (1963). Situations, actions, and causal laws. Memo 2, Stanford University Artificial Intelligence Project, Stanford, California.

McCarthy, J. (1968). Programs with common sense. In Minsky, M. L. (Ed.), *Semantic Information Processing*, pp. 403–418. MIT Press, Cambridge, Massachusetts.

McCarthy, J. (1980). Circumscription: A form of non-monotonic reasoning. *Artificial Intelligence*, *13*(1–2), 27–39.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D., and Swann, M. (Eds.), *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press, Edinburgh, Scotland.

McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (1955). Proposal for the Dartmouth summer research project on artificial intelligence. Tech. rep., Dartmouth College.

McCawley, J. D. (1988). *The Syntactic Phenomena of English*, Vol. 2 volumes. University of Chicago Press.

McCawley, J. D. (1993). *Everything That Linguists Have Always Wanted to Know about Logic but Were Ashamed to Ask* (Second edition). University of Chicago Press, Chicago.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–137.

McCune, W. (1992). Automated discovery of new axiomatizations of the left group and right group calculi. *Journal of Automated Reasoning*, *9*(1), 1–24.

McCune, W. (1997). Solution of the robbins problem. *Journal of Automated Reasoning*, *19*(3), 263–276.

McDermott, D. (1976). Artificial intelligence meets natural stupidity. *SIGART Newsletter*, *57*, 4–9.

McDermott, D. (1978a). Planning and acting. *Cognitive Science*, *2*(2), 71–109.

McDermott, D. (1978b). Tarskian semantics, or, no notation without denotation!. *Cognitive Science*, *2*(3).

McDermott, D. (1987). A critique of pure reason. *Computational Intelligence*, *3*(3), 151–237.

McDermott, D. (1996). A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on AI Planning Systems*, pp. 142–149, Edinburgh, Scotland. AAAI Press.

McDermott, D. and Doyle, J. (1980). Non-monotonic logic: i. *Artificial Intelligence*, *13*(1–2), 41–72.

McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence*, *19*(1), 39–88.

McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998). Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, *16*(2), 140–152.

McGregor, J. J. (1979). Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, *19*(3), 229–250.

McKeown, K. (1985). *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, Cambridge, UK.

McLachlan, G. J. and Krishnan, T. (1997). *The EM Algorithm and Extensions*. Wiley, New York.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer, Dordrecht, Netherlands.

Meehl, P. (1955). *Clinical vs. Statistical Prediction*. University of Minnesota Press, Minneapolis.

Melćuk, I. A. and Polguere, A. (1988). A formal lexicon in the meaning-text theory (or how to do lexica with words). *Computational Linguistics*, *13*(3–4), 261–275.

Mendel, G. (1866). Versuche über pflanzen-hybriden. *Verhandlungen des Naturforschenden Vereins, Abhandlungen, Brünn*, *4*, 3–47. Translated into English by C. T. Druery, published by Bateson (1902).

Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London, A*, *209*, 415–446.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, *21*, 1087–1091.

Mézard, M. and Nadal, J.-P. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics*, *22*, 2191–2204.

Michalski, R. S. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the First International Symposium on Information Processing*, pp. 125–128.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds.). (1983). *Machine Learning: An Artificial Intelligence Approach*, Vol. 1. Morgan Kaufmann, San Mateo, California.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds.). (1986a). *Machine Learning: An Artificial Intelligence Approach*, Vol. 2. Morgan Kaufmann, San Mateo, California.

Michalski, R. S., Mozetic, I., Hong, J., and Lavrač, N. (1986b). The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pp. 1041–1045, Philadelphia. Morgan Kaufmann.

Michel, S. and Plamondon, P. (1996). Bilingual sentence alignment: Balancing robustness and accuracy. In *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA)*.

Michie, D. (1966). Game-playing and game-learning automata. In Fox, L. (Ed.), *Advances in Programming and Non-Numerical Computation*, pp. 183–200. Pergamon, Oxford, UK.

Michie, D. (1972). Machine intelligence at Edinburgh. *Management Informatics*, *2*(1), 7–12.

Michie, D. (1974). Machine intelligence at Edinburgh. In *On Intelligence*, pp. 143–155. Edinburgh University Press.

Michie, D. and Chambers, R. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. and Michie, D. (Eds.), *Machine Intelligence 2*, pp. 125–133. Elsevier/North-Holland, Amsterdam, London, New York.

Michie, D., Spiegelhalter, D. J., and Taylor, C. (Eds.). (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Chichester, England.

Milgrom, P. (1997). Putting auction theory to work: The simultaneous ascending auction. Tech. rep. Technical Report 98-0002, Stanford University Department of Economics.

Mill, J. S. (1843). *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence, and Methods of Scientific Investigation*. J. W. Parker, London.

Mill, J. S. (1863). *Utilitarianism*. Parker, Son and Bourn, London.

Miller, A. C., Merkhofer, M. M., Howard, R. A., Matheson, J. E., and Rice, T. R. (1976). Development of automated aids for decision analysis. Technical report, SRI International, Menlo Park, California.

Minsky, M. L. (Ed.). (1968). *Semantic Information Processing*. MIT Press, Cambridge, Massachusetts.

Minsky, M. L. (1975). A framework for representing knowledge. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*, pp. 211–277. McGraw-Hill, New York. Originally an MIT AI Laboratory memo; the 1975 version is abridged, but is the most widely cited.

Minsky, M. L. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry* (first edition). MIT Press, Cambridge, Massachusetts.

Minsky, M. L. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry* (Expanded edition). MIT Press, Cambridge, Massachusetts.

Minton, S. (1984). Constraint-based generalization: Learning game-playing plans from single examples. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, pp. 251–254, Austin, Texas. Morgan Kaufmann.

Minton, S. (1988). Quantitative results concerning the utility of explanation- based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pp. 564–569, St. Paul, Minnesota. Morgan Kaufmann.

Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, *58*(1–3), 161–205.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, Massachusetts.

Mitchell, M., Holland, J. H., and Forrest, S. (1996). When will a genetic algorithm outperform hill climbing?. In Cowan, J., Tesauro, G., and Alspector, J. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 6. MIT Press, Cambridge, Massachusetts.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pp. 305–310, Cambridge, Massachusetts. IJCAII.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*(2), 203–226.

Mitchell, T. M. (1990). Becoming increasingly reactive (mobile robots). In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, pp. 1051–1058, Boston. MIT Press.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Mitchell, T. M., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, *1*, 47–80.

Mitchell, T. M., Utgoff, P. E., and Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, pp. 163–190. Morgan Kaufmann, San Mateo, California.

Mitkov, R. (2002). *Anaphora Resolution*. Longman, New York.

Mohr, R. and Henderson, T. C. (1986). Arc and path consistency revisited. *Artificial Intelligence*, *28*(2), 225–233.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, *16*(1), 69–88.

Montague, P. R., Dayan, P., Person, C., and Sejnowski, T. (1995). Bee foraging in uncertain environments using predictive Hebbian learning. *Nature*, *377*, 725–728.

Montague, R. (1970). English as a formal language. In *Linguaggi nella Società e nella Tecnica*, pp. 189–224. Edizioni di Comunità, Milan.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Hintikka, K. J. J., Moravcsik, J. M. E., and Suppes, P. (Eds.), *Approaches to Natural Language*. D. Reidel, Dordrecht, Netherlands.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, *7*(2), 95–132.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, Edmonton, Alberta. AAAI Press.

Mooney, R. (1999). Learning for semantic interpretation: Scaling up without dumbing down. In Cussens, J. (Ed.), *Proceedings of the 1st Workshop on Learning Language in Logic*, pp. 7–15. Springer-Verlag.

Mooney, R. J. and Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of AI Research*, *3*, 1–24.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping—reinforcement learning with less data and less time. *Machine Learning*, *13*, 103–130.

Moore, E. F. (1959). The shortest path through a maze. In *Proceedings of an International Symposium on the Theory of Switching, Part II*, pp. 285–292. Harvard University Press, Cambridge, Massachusetts.

Moore, J. S. and Newell, A. (1973). How can Merlin understand?. In Gregg, L. (Ed.), *Knowledge and Cognition*. Lawrence Erlbaum Associates, Potomac, Maryland.

Moore, R. C. (1980). Reasoning about knowledge and action. Artificial intelligence center technical note 191, SRI International, Menlo Park, California.

Moore, R. C. (1985). A formal theory of knowledge and action. In Hobbs, J. R. and Moore, R. C. (Eds.), *Formal Theories of the Commonsense World*, pp. 319–358. Ablex, Norwood, New Jersey.

Moravec, H. P. (1983). The stanford cart and the cmu rover. *Proceedings of the IEEE*, *71*(7), 872–884.

Moravec, H. P. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation*, pp. 116–121, St. Louis, Missouri. IEEE Computer Society Press.

Moravec, H. P. (1988). *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, Cambridge, Massachusetts.

Moravec, H. P. (2000). *Robot: Mere Machine to Transcendent Mind*. Oxford University Press.

Morgenstern, L. (1987). Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 867–874, Milan. Morgan Kaufmann.

Morgenstern, L. (1998). Inheritance comes of age: Applying nonmonotonic techniques to problems in industry. *Artificial Intelligence*, *103*, 237–271.

Morjaria, M. A., Rink, F. J., Smith, W. D., Klempner, G., Burns, C., and Stein, J. (1995). Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 141–148. Morgan Kaufmann.

Morrison, P. and Morrison, E. (Eds.). (1961). *Charles Babbage and His Calculating Engines: Selected Writings by Charles Babbage and Others*. Dover, New York.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pp. 530–535. ACM Press.

Mosteller, F. and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Addison-Wesley.

Mostow, J. and Prieditis, A. E. (1989). Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 1, pp. 701–707, Detroit. Morgan Kaufmann.

Motzkin, T. S. and Schoenberg, I. J. (1954). The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, *6*(3), 393–404.

Moussouris, J., Holloway, J., and Greenblatt, R. D. (1979). CHEOPS: A chess-oriented processing system. In Hayes, J. E., Michie, D., and Mikulich, L. I. (Eds.), *Machine Intelligence 9*, pp. 351–360. Ellis Horwood, Chichester, England.

Moutarlier, P. and Chatila, R. (1989). Stochastic multisensory data fusion for mobile robot location and environment modeling. In *5th Int. Symposium on Robotics Research*, Tokyo.

Muggleton, S. H. (1991). Inductive logic programming. *New Generation Computing*, *8*, 295–318.

Muggleton, S. H. (1992). *Inductive Logic Programming*. Academic Press, New York.

Muggleton, S. H. (1995). Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, *13*(3-4), 245–286.

Muggleton, S. H. (2000). Learning stochastic logic programs. Proceedings of the AAAI 2000 Workshop on Learning Statistical Models from Relational Data.

Muggleton, S. H. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pp. 339–352. Morgan Kaufmann.

Muggleton, S. H. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19/20*, 629–679.

Muggleton, S. H. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the Workshop on Algorithmic Learning Theory*, pp. 368–381, Tokyo. Ohmsha.

Müller, M. (2002). Computer Go. *Artificial Intelligence*, *134*(1–2), 145–179.

Mundy, J. and Zisserman, A. (Eds.). (1992). *Geometric Invariance in Computer Vision*. MIT Press, Cambridge, Massachusetts.

Murphy, K., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, pp. 467–475, Stockholm. Morgan Kaufmann.

Murphy, K. and Russell, S. J. (2001). Rao-blackwellised particle filtering for dynamic bayesian networks. In Doucet, A., de Freitas, N., and Gordon, N. J. (Eds.), *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, Berlin.

Murphy, R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, Massachusetts.

Muscettola, N., Nayak, P., Pell, B., and Williams, B. (1998). Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, *103*, 5–48.

Myerson, R. B. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge.

Nagel, T. (1974). What is it like to be a bat?. *Philosophical Review*, *83*, 435–450.

Nalwa, V. S. (1993). *A Guided Tour of Computer Vision*. Addison-Wesley, Reading, Massachusetts.

Nash, J. (1950). Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the United States of America*, *36*, 48–49.

Nau, D. S. (1980). Pathology on game trees: A summary of results. In *Proceedings of the First Annual National Conference on Artificial Intelligence (AAAI-80)*, pp. 102–104, Stanford, California. AAAI.

Nau, D. S. (1983). Pathology on game trees revisited, and an alternative to minimaxing. *Artificial Intelligence*, *21*(1–2), 221–244.

Nau, D. S., Kumar, V., and Kanal, L. N. (1984). General branch and bound, and its relation to A* and AO*. *Artificial Intelligence*, *23*, 29–58.

Naur, P. (1963). Revised report on the algorithmic language Algol 60. *Communications of the Association for Computing Machinery*, *6*(1), 1–17.

Nayak, P. and Williams, B. (1997). Fast context switching in real-time propositional reasoning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 50–56, Providence, Rhode Island. AAAI Press.

Neal, R. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin.

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research*, *12*, 271–315.

Nelson, G. and Oppen, D. C. (1979). Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, *1*(2), 245–257.

Netto, E. (1901). *Lehrbuch der Combinatorik*. B. G. Teubner, Leipzig.

Nevill-Manning, C. G. and Witten, I. H. (1997). Identifying hierarchical structures in sequences: A linear-time algorithm. *Journal of AI Research*, *7*, 67–82.

Nevins, A. J. (1975). Plane geometry theorem proving using forward chaining. *Artificial Intelligence*, *6*(1), 1–23.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, *18*(1), 82–127.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts.

Newell, A. and Ernst, G. (1965). The search for generality. In Kalenich, W. A. (Ed.), *Information Processing 1965: Proceedings of IFIP Congress 1965*, Vol. 1, pp. 17–24, Chicago. Spartan.

Newell, A., Shaw, J. C., and Simon, H. A. (1957). Empirical explorations with the logic theory machine. *Proceedings of the Western Joint Computer Conference*, *15*, 218–239. Reprinted in Feigenbaum and Feldman (1963).

Newell, A., Shaw, J. C., and Simon, H. A. (1958). Chess playing programs and the problem of complexity. *IBM Journal of Research and Development*, *4*(2), 320–335.

Newell, A. and Simon, H. A. (1961). GPS, a program that simulates human thought. In Billing, H. (Ed.), *Lernende Automaten*, pp. 109–124. R. Oldenbourg, Munich.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall, Upper Saddle River, New Jersey.

Newell, A. and Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the Association for Computing Machinery*, *19*, 113–126.

Newton, I. (1664–1671). Methodus fluxionum et serierum infinitarum. Unpublished notes.

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia. Morgan Kaufmann.

Ng, A. Y. and Jordan, M. I. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference*, pp. 406–415, Stanford, California. Morgan Kaufmann.

Nguyen, X. and Kambhampati, S. (2001). Reviving partial order planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 459–466, Seattle. Morgan Kaufmann.

Nguyen, X., Kambhampati, S., and Nigenda, R. S. (2001). Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. Tech. rep., Computer Science and Engineering Department, Arizona State University.

Nicholson, A. and Brady, J. M. (1992). The data association problem when monitoring robot vehicles using dynamic belief networks. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pp. 689–693, Vienna, Austria. Wiley.

Niemelä, I., Simons, P., and Syrjänen, T. (2000). Smodels: A system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*.

Nilsson, D. and Lauritzen, S. (2000). Evaluating influence diagrams using LIMIDs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference*, pp. 436–445, Stanford, California. Morgan Kaufmann.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems.* McGraw-Hill, New York. republished in 1990.

Nilsson, N. J. (1971). *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence.* Morgan Kaufmann, San Mateo, California.

Nilsson, N. J. (1984). Shakey the robot. Technical note 323, SRI International, Menlo Park, California.

Nilsson, N. J. (1986). Probabilistic logic. *Artificial Intelligence*, *28*(1), 71–87.

Nilsson, N. J. (1991). Logic and artificial intelligence. *Artificial Intelligence*, *47*(1–3), 31–56.

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann, San Mateo, California.

Norvig, P. (1988). Multiple simultaneous interpretations of ambiguous sentences. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society.*

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp.* Morgan Kaufmann, San Mateo, California.

Nowick, S. M., Dean, M. E., Dill, D. L., and Horowitz, M. (1993). The design of a high-performance cache controller: A case study in asynchronous synthesis. *Integration: The VLSI Journal*, *15*(3), 241–262.

Nunberg, G. (1979). The non-uniqueness of semantic solutions: Polysemy. *Language and Philosophy*, *3*(2), 143–184.

Nussbaum, M. C. (1978). *Aristotle's* De Motu Animalium. Princeton University Press, Princeton, New Jersey.

Ogawa, S., Lee, T.-M., Kay, A. R., and Tank, D. W. (1990). Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *Proceedings of the National Academy of Sciences of the United States of America*, *87*, 9868–9872.

Olawsky, D. and Gini, M. (1990). Deferred planning and sensor use. In Sycara, K. P. (Ed.), *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, California. Defense Advanced Research Projects Agency (DARPA), Morgan Kaufmann.

Olesen, K. G. (1993). Causal probabilistic networks with both discrete and continuous variables. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *15*(3), 275–279.

Oliver, R. M. and Smith, J. Q. (Eds.). (1990). *Influence Diagrams, Belief Nets and Decision Analysis.* Wiley, New York.

Olson, C. F. (1994). Time and space efficient pose clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 251–258, Washington, DC. IEEE Computer Society Press.

Oncina, J. and Garcia, P. (1992). Inferring regular languages in polynomial update time. In Perez, Sanfeliu, and Vidal (Eds.), *Pattern Recognition and Image Analysis*, pp. 49–61. World Scientific.

O'Reilly, U.-M. and Oppacher, F. (1994). Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Davidor, Y., Schwefel, H.-P., and Manner, R. (Eds.), *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pp. 397–406, Jerusalem, Israel. Springer-Verlag.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, *49*(2–3), 161–178.

Ortony, A. (Ed.). (1979). *Metaphor and Thought.* Cambridge University Press, Cambridge, UK.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory.* MIT Press, Cambridge, Massachusetts.

Osherson, D. N., Stob, M., and Weinstein, S. (1986). *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists.* MIT Press, Cambridge, Massachusetts.

Page, C. D. and Srinivasan, A. (2002). ILP: A short look back and a longer look forward. Submitted to Journal of Machine Learning Research.

Pak, I. (2001). On mixing of certain random walks, cutoff phenomenon and sharp threshold of random matroid processes. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, *110*, 251–272.

Palay, A. J. (1985). *Searching with Probabilities.* Pitman, London.

Palmer, D. A. and Hearst, M. A. (1994). Adaptive sentence boundary disambiguation. In *Proceedings of the Conference on Applied Natural Language Processing*, pp. 78–83. Morgan Kaufmann.

Palmer, S. (1999). *Vision Science: Photons to Phenomenology.* MIT Press, Cambridge, Massachusetts.

Papadimitriou, C. H. (1994). *Computational Complexity.* Addison Wesley.

Papadimitriou, C. H., Tamaki, H., Raghavan, P., and Vempala, S. (1998). Latent semantic indexing: A probabilistic analysis. In *Proceedings of the ACM Conference on Principles of Database Systems (PODS)*, pp. 159–168, New York. ACM Press.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Papadimitriou, C. H. and Yannakakis, M. (1991). Shortest paths without a map. *Theoretical Computer Science*, *84*(1), 127–150.

Papavassiliou, V. and Russell, S. J. (1999). Convergence of reinforcement learning with general function approximators. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 748–757, Stockholm. Morgan Kaufmann.

Parekh, R. and Honavar, V. (2001). Dfa learning from simple examples. *Machine Learning*, *44*, 9–35.

Parisi, G. (1988). *Statistical field theory*. Addison-Wesley, Reading, Massachusetts.

Parker, D. B. (1985). Learning logic. Technical report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Parker, L. E. (1996). On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, *10*(6).

Parr, R. and Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In Jordan, M. I., Kearns, M., and Solla, S. A. (Eds.), *Advances in Neural Information Processing Systems 10*. MIT Press, Cambridge, Massachusetts.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, *33*, 1065–1076.

Pasula, H. and Russell, S. J. (2001). Approximate inference for first-order probabilistic languages. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle. Morgan Kaufmann.

Pasula, H., Russell, S. J., Ostland, M., and Ritov, Y. (1999). Tracking many objects with many sensors. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm. Morgan Kaufmann.

Paterson, M. S. and Wegman, M. N. (1978). Linear unification. *Journal of Computer and System Sciences*, *16*, 158–167.

Patrick, B. G., Almulla, M., and Newborn, M. M. (1992). An upper bound on the time complexity of iterative-deepening-A*. *Annals of Mathematics and Artificial Intelligence*, *5*(2–4), 265–278.

Patten, T. (1988). *Systemic Text Generation as Problem Solving*. Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK.

Paul, R. P. (1981). *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, Massachusetts.

Peano, G. (1889). *Arithmetices principia, nova methodo exposita*. Fratres Bocca, Turin.

Pearl, J. (1982a). Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pp. 133–136, Pittsburgh, Pennsylvania. Morgan Kaufmann.

Pearl, J. (1982b). The solution for the branching factor of the alpha–beta pruning algorithm and its optimality. *Communications of the Association for Computing Machinery*, *25*(8), 559–564.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, Massachusetts.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, *29*, 241–288.

Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, *32*, 247–257.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, UK.

Pearl, J. and Verma, T. (1991). A theory of inferred causation. In Allen, J. A., Fikes, R., and Sandewall, E. (Eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 441–452, San Mateo, California. Morgan Kaufmann.

Pearson, J. and Jeavons, P. (1997). A survey of tractable constraint satisfaction problems. Technical report CSD-TR-97-15, Royal Holloway College, U. of London.

Pednault, E. P. D. (1986). Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeff, M. P. and Lansky, A. L. (Eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pp. 47–82, Timberline, Oregon. Morgan Kaufmann.

Peirce, C. S. (1870). Description of a notation for the logic of relatives, resulting from an amplification of the conceptions of Boole's calculus of logic. *Memoirs of the American Academy of Arts and Sciences*, *9*, 317–378.

Peirce, C. S. (1883). A theory of probable inference. Note B. The logic of relatives. In *Studies in Logic by Members of the Johns Hopkins University*, pp. 187–203, Boston.

Peirce, C. S. (1902). Logic as semiotic: The theory of signs. Unpublished manuscript; reprinted in (Buchler 1955).

Peirce, C. S. (1909). Existential graphs. Unpublished manuscript; reprinted in (Buchler 1955).

Pelikan, M., Goldberg, D. E., and Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 525–532, Orlando, Florida. Morgan Kaufmann.

Pemberton, J. C. and Korf, R. E. (1992). Incremental planning on graphs with cycles. In Hendler, J. (Ed.), *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pp. 525–532, College Park, Maryland. Morgan Kaufmann.

Penberthy, J. S. and Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, pp. 103–114. Morgan Kaufmann.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, *2*, 437–454.

Penrose, R. (1989). *The Emperor's New Mind*. Oxford University Press, Oxford, UK.

Penrose, R. (1994). *Shadows of the Mind*. Oxford University Press, Oxford, UK.

Peot, M. and Smith, D. E. (1992). Conditional nonlinear planning. In Hendler, J. (Ed.), *Proceedings of the First International Conference on AI Planning Systems*, pp. 189–197, College Park, Maryland. Morgan Kaufmann.

Pereira, F. and Shieber, S. M. (1987). *Prolog and Natural-Language Analysis*. Center for the Study of Language and Information (CSLI), Stanford, California.

Pereira, F. and Warren, D. H. D. (1980). Definite clause grammars for language analysis: A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, *13*, 231–278.

Peterson, C. and Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, *1*(5), 995–1019.

Pfeffer, A. (2000). *Probabilistic Reasoning for Complex Systems*. Ph.D. thesis, Stanford University, Stanford, California.

Pinker, S. (1989). *Learnability and Cognition*. MIT Press, Cambridge, MA.

Pinker, S. (1995). Language acquisition. In Gleitman, L. R., Liberman, M., and Osherson, D. N. (Eds.), *An Invitation to Cognitive Science* (second edition)., Vol. 1. MIT Press, Cambridge, Massachusetts.

Pinker, S. (2000). *The Language Instinct: How the Mind Creates Language*. MIT Press, Cambridge, Massachusetts.

Plaat, A., Schaeffer, J., Pijls, W., and de Bruin, A. (1996). Best-first fixed-depth minimax algorithms. *Artificial Intelligence Journal*, *87*(1–2), 255–293.

Place, U. T. (1956). Is consciousness a brain process?. *British Journal of Psychology*, *47*, 44–50.

Plotkin, G. (1971). *Automatic Methods of Inductive Inference*. Ph.D. thesis, Edinburgh University.

Plotkin, G. (1972). Building-in equational theories. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 7*, pp. 73–90. Edinburgh University Press, Edinburgh, Scotland.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pp. 46–57, Providence, Rhode Island. IEEE, IEEE Computer Society Press.

Pohl, I. (1969). Bi-directional and heuristic search in path problems. Tech. rep. 104, SLAC (Stanford Linear Accelerator Center, Stanford, California.

Pohl, I. (1970). First results on the effect of error in heuristic search. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 5*, pp. 219–236. Elsevier/North-Holland, Amsterdam, London, New York.

Pohl, I. (1971). Bi-directional search. In Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence 6*, pp. 127–140. Edinburgh University Press, Edinburgh, Scotland.

Pohl, I. (1973). The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pp. 20–23, Stanford, California. IJCAII.

Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. In Elcock, E. W. and Michie, D. (Eds.), *Machine Intelligence 8*, pp. 55–72. Ellis Horwood, Chichester, England.

Pomerleau, D. A. (1993). *Neural Network Perception for Mobile Robot Guidance*. Kluwer, Dordrecht, Netherlands.

Ponte, J. M. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Research and Development in Information Retrieval*, pp. 275–281.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence, 64*, 81–129.

Poole, D., Mackworth, A. K., and Goebel, R. (1998). *Computational intelligence: A logical approach*. Oxford University Press, Oxford, UK.

Popper, K. R. (1959). *The Logic of Scientific Discovery*. Basic Books, New York.

Popper, K. R. (1962). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Basic Books, New York.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program, 13*(3), 130–137.

Post, E. L. (1921). Introduction to a general theory of elementary propositions. *American Journal of Mathematics, 43*, 163–185.

Pradhan, M., Provan, G. M., Middleton, B., and Henrion, M. (1994). Knowledge engineering for large belief networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, pp. 484–490, Seattle, Washington. Morgan Kaufmann.

Pratt, V. R. (1976). Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th IEEE Symposium on the Foundations of Computer Science*, pp. 109–121. IEEE Computer Society Press.

Prawitz, D. (1960). An improved proof procedure. *Theoria, 26*, 102–139.

Prawitz, D. (1965). *Natural Deduction: A Proof Theoretical Study*. Almquist and Wiksell, Stockholm.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2002). *Numerical Recipes in C++: The Art of Scientific Computing* (Second edition). Cambridge University Press, Cambridge, UK.

Prieditis, A. E. (1993). Machine discovery of effective admissible heuristics. *Machine Learning, 12*(1–3), 117–141.

Prinz, D. G. (1952). Robot chess. *Research, 5*, 261–266.

Prior, A. N. (1967). *Past, Present, and Future*. Oxford University Press, Oxford, UK.

Prosser, P. (1993). Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence, 9*, 268–299.

Pryor, L. and Collins, G. (1996). Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research, 4*, 287–339.

Pullum, G. K. (1991). *The Great Eskimo Vocabulary Hoax (and Other Irreverent Essays on the Study of Language)*. University of Chicago Press, Chicago.

Pullum, G. K. (1996). Learnability, hyperlearning, and the poverty of the stimulus. In *22nd Annual Meeting of the Berkeley Linguistics Society*.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York.

Puterman, M. L. and Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science, 24*(11), 1127–1137.

Putnam, H. (1960). Minds and machines. In Hook, S. (Ed.), *Dimensions of Mind*, pp. 138–164. Macmillan, London.

Putnam, H. (1963). 'Degree of confirmation' and inductive logic. In Schilpp, P. A. (Ed.), *The Philosophy of Rudolf Carnap*, pp. 270–292. Open Court, La Salle, Illinois.

Putnam, H. (1967). The nature of mental states. In Capitan, W. H. and Merrill, D. D. (Eds.), *Art, Mind, and Religion*, pp. 37–48. University of Pittsburgh Press, Pittsburgh.

Pylyshyn, Z. W. (1974). Minds, machines and phenomenology: Some reflections on Dreyfus' "What Computers Can't Do". *International Journal of Cognitive Psychology, 3*(1), 57–77.

Pylyshyn, Z. W. (1984). *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press, Cambridge, Massachusetts.

Quillian, M. R. (1961). A design for an understanding machine. Paper presented at a colloquium: Semantic Problems in Natural Language, King's College, Cambridge, England.

Quine, W. V. (1953). Two dogmas of empiricism. In *From a Logical Point of View*, pp. 20–46. Harper and Row, New York.

Quine, W. V. (1960). *Word and Object*. MIT Press, Cambridge, Massachusetts.

Quine, W. V. (1982). *Methods of Logic* (Fourth edition). Harvard University Press, Cambridge, Massachusetts.

Quinlan, E. and O'Brien, S. (1992). Sublanguage: Characteristics and selection guidelines for MT. In *AI and Cognitive Science '92: Proceedings of Annual Irish Conference on Artificial Intelligence and Cognitive Science '92*, pp. 342–345, Limerick, Ireland. Springer-Verlag.

Quinlan, J. R. (1979). Discovering rules from large collections of examples: A case study. In Michie, D. (Ed.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, California.

Quinlan, J. R. and Cameron-Jones, R. M. (1993). FOIL: a midterm report. In Brazdil, P. B. (Ed.), *European Conference on Machine Learning Proceedings (ECML-93)*, pp. 3–20, Vienna. Springer-Verlag.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman, New York.

Rabani, Y., Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. *Random Structures and Algorithms*, *12*(4), 313–334.

Rabiner, L. R. and Juang, B.-H. (1993). *Fundamentals of Speech Recognition*. Prentice-Hall, Upper Saddle River, New Jersey.

Ramakrishnan, R. and Ullman, J. D. (1995). A survey of research in deductive database systems. *Journal of Logic Programming*, *23*(2), 125–149.

Ramsey, F. P. (1931). Truth and probability. In Braithwaite, R. B. (Ed.), *The Foundations of Mathematics and Other Logical Essays*. Harcourt Brace Jovanovich, New York.

Raphson, J. (1690). *Analysis aequationum universalis*. Apud Abelem Swalle, London.

Rassenti, S., Smith, V., and Bulfin, R. (1982). A combinatorial auction mechanism for airport time slot allocation.. *Bell Journal of Economics*, *13*, 402–417.

Ratner, D. and Warmuth, M. (1986). Finding a shortest solution for the $n \times n$ extension of the 15-puzzle is intractable. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 168–172, Philadelphia. Morgan Kaufmann.

Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, *3*(8), 1445–1450.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Library translation 1122, Royal Aircraft Establishment.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany.

Regin, J. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 362–367, Seattle. AAAI Press.

Reichenbach, H. (1949). *The Theory of Probability: An Inquiry into the Logical and Mathematical Foundations of the Calculus of Probability* (Second edition). University of California Press, Berkeley and Los Angeles.

Reif, J. (1979). Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pp. 421–427, San Juan, Puerto Rico. IEEE, IEEE Computer Society Press.

Reiter, E. and Dale, R. (2000). *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, *13*(1–2), 81–132.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V. (Ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press, New York.

Reiter, R. (2001a). On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, *2*(4), 433–457.

Reiter, R. (2001b). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, Massachusetts.

Reitman, W. and Wilcox, B. (1979). The structure and performance of the INTERIM.2 Go program. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, pp. 711–719, Tokyo. IJCAII.

Remus, H. (1962). Simulation of a learning machine for playing Go. In *Proceedings IFIP Congress*, pp. 428–432, Amsterdam, London, New York. Elsevier/North-Holland.

Rényi, A. (1970). *Probability Theory*. Elsevier/North-Holland, Amsterdam, London, New York.

Rescher, N. and Urquhart, A. (1971). *Temporal Logic*. Springer-Verlag, Berlin.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, *21*, 25–34. SIGGRAPH '87 Conference Proceedings.

Rich, E. and Knight, K. (1991). *Artificial Intelligence* (second edition). McGraw-Hill, New York.

Richardson, M., Bilmes, J., and Diorio, C. (2000). Hidden-articulator Markov models: Performance improvements and robustness to noise. In *ICASSP-2000: 2000 International Conference on Acoustics, Speech, and Signal Processing*, Los Alamitos, CA. IEEE Computer Society Press.

Rieger, C. (1976). An organization of knowledge for problem solving and language comprehension. *Artificial Intelligence*, *7*, 89–127.

Ringle, M. (1979). *Philosophical Perspectives in Artificial Intelligence*. Humanities Press, Atlantic Highlands, New Jersey.

Rintanen, J. (1999). Improvements to the evaluation of quantified boolean formulae. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 1192–1197, Stockholm. Morgan Kaufmann.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK.

Rissanen, J. (1984). Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, *IT-30*(4), 629–636.

Ritchie, G. D. and Hanna, F. K. (1984). AM: A case study in AI methodology. *Artificial Intelligence*, *23*(3), 249–268.

Rivest, R. (1987). Learning decision lists. *Machine Learning*, *2*(3), 229–246.

Roberts, L. G. (1963). Machine perception of three-dimensional solids. Technical report 315, MIT Lincoln Laboratory.

Robertson, N. and Seymour, P. D. (1986). Graph minors. ii. Algorithmic aspects of tree-width. *Journal of Algorithms*, *7*(3), 309–322.

Robertson, S. E. (1977). The probability ranking principle in ir. *Journal of Documentation*, *33*, 294–304.

Robertson, S. E. and Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, *27*, 129–146.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, *12*, 23–41.

Roche, E. and Schabes, Y. (1997). *Finite-State Language Processing (Language, Speech and Communication)*. Bradford Books, Cambridge.

Rock, I. (1984). *Perception*. W. H. Freeman, New York.

Rorty, R. (1965). Mind-body identity, privacy, and categories. *Review of Metaphysics*, *19*(1), 24–54.

Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton. Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Ithaca, New York.

Rosenblatt, F. (1960). On the convergence of reinforcement procedures in simple perceptrons. Report VG-1196-G-4, Cornell Aeronautical Laboratory, Ithaca, New York.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, Chicago.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, *27*, 832–837.

Rosenblueth, A., Wiener, N., and Bigelow, J. (1943). Behavior, purpose, and teleology. *Philosophy of Science*, *10*, 18–24.

Rosenschein, J. S. and Zlotkin, G. (1994). *Rules of Encounter*. MIT Press, Cambridge, Massachusetts.

Rosenschein, S. J. (1985). Formal theories of knowledge in AI and robotics. *New Generation Computing*, *3*(4), 345–357.

Rosenthal, D. M. (Ed.). (1971). *Materialism and the Mind-Body Problem*. Prentice-Hall, Upper Saddle River, New Jersey.

Ross, S. M. (1988). *A First Course in Probability* (third edition). Macmillan, London.

Roussel, P. (1975). Prolog: Manual de reference et d'utilization. Tech. rep., Groupe d'Intelligence Artificielle, Université d'Aix-Marseille.

Rouveirol, C. and Puget, J.-F. (1989). A simple and general solution for inverting resolution. In *Proceedings of the European Working Session on Learning*, pp. 201–210, Porto, Portugal. Pitman.

Rowat, P. F. (1979). *Representing the Spatial Experience and Solving Spatial problems in a Simulated Robot Environment*. Ph.D. thesis, University of British Columbia, Vancouver, BC, Canada.

Roweis, S. T. and Ghahramani, Z. (1999). A unifying review of Linear Gaussian Models. *Neural Computation*, *11*(2), 305–345.

Rubin, D. (1988). Using the SIR algorithm to simulate posterior distributions. In Bernardo, J. M., de Groot, M. H., Lindley, D. V., and Smith, A. F. M. (Eds.), *Bayesian Statistics 3*, pp. 395–402. Oxford University Press, Oxford, UK.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, Vol. 1, chap. 8, pp. 318–362. MIT Press, Cambridge, Massachusetts.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Rumelhart, D. E. and McClelland, J. L. (Eds.). (1986). *Parallel Distributed Processing*. MIT Press, Cambridge, Massachusetts.

Ruspini, E. H., Lowrance, J. D., and Strat, T. M. (1992). Understanding evidential reasoning. *International Journal of Approximate Reasoning*, *6*(3), 401–424.

Russell, J. G. B. (1990). Is screening for abdominal aortic aneurysm worthwhile?. *Clinical Radiology*, *41*, 182–184.

Russell, S. J. (1985). The compleat guide to MRS. Report STAN-CS-85-1080, Computer Science Department, Stanford University.

Russell, S. J. (1986). A quantitative analysis of analogy by similarity. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pp. 284–288, Philadelphia. Morgan Kaufmann.

Russell, S. J. (1988). Tree-structured bias. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Vol. 2, pp. 641–645, St. Paul, Minnesota. Morgan Kaufmann.

Russell, S. J. (1992). Efficient memory-bounded search methods. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pp. 1–5, Vienna. Wiley.

Russell, S. J. (1998). Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual ACM Workshop on Computational Learning Theory (COLT-98)*, pp. 101–103, Madison, Wisconsin. ACM Press.

Russell, S. J., Binder, J., Koller, D., and Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1146–52, Montreal. Morgan Kaufmann.

Russell, S. J. and Grosof, B. (1987). A declarative approach to bias in concept learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle. Morgan Kaufmann.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey.

Russell, S. J. and Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, *3*, 575–609.

Russell, S. J., Subramanian, D., and Parr, R. (1993). Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 338–345, Chambery, France. Morgan Kaufmann.

Russell, S. J. and Wefald, E. H. (1989). On optimal game-tree search using rational meta-reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 334–340, Detroit. Morgan Kaufmann.

Russell, S. J. and Wefald, E. H. (1991). *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, Massachusetts.

Rustagi, J. S. (1976). *Variational Methods in Statistics*. Academic Press, New York.

Ryder, J. L. (1971). Heuristic analysis of large trees as generated in the game of Go. Memo AIM-155, Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Stanford, California.

Sabin, D. and Freuder, E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. In *ECAI 94: 11th European Conference on Artificial Intelligence. Proceedings*, pp. 125–129, Amsterdam. Wiley.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, *5*(2), 115–135.

Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pp. 206–214, Tbilisi, Georgia. IJCAII.

Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. Elsevier/North-Holland, Amsterdam, London, New York.

Sacerdoti, E. D., Fikes, R. E., Reboh, R., Sagalowicz, D., Waldinger, R., and Wilber, B. M. (1976). QLISP—a language for the interactive development of complex systems. In *Proceedings of the AFIPS National Computer Conference*, pp. 349–356.

Sacks, E. and Joskowicz, L. (1993). Automated modeling and kinematic simulation of mechanisms. *Computer Aided Design*, *25*(2), 106–118.

Sadri, F. and Kowalski, R. (1995). Variants of the event calculus. In *International Conference on Logic Programming*, pp. 67–81.

Sag, I. and Wasow, T. (1999). *Syntactic Theory: An Introduction*. CSLI Publications, Stanford, California.

Sager, N. (1981). *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Massachusetts.

Sahami, M., Dumais, S. T., Heckerman, D., and Horvitz, E. J. (1998). A Bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin. AAAI Technical Report WS-98-05.

Sahami, M., Hearst, M. A., and Saund, E. (1996). Applying the multiple cause mixture model to text categorization. In Saitta, L. (Ed.), *Proceedings of ICML-96, 13th International Conference on Machine Learning*, pp. 435–443, Bari, Italy. Morgan Kaufmann Publishers.

Salomaa, A. (1969). Probabilistic and weighted grammars. *Information and Control*, *15*, 529–544.

Salton, G. and McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, *18*(11), 613–620.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 210–229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers II—Recent progress. *IBM Journal of Research and Development*, *11*(6), 601–617.

Samuelsson, C. and Rayner, M. (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 609–615, Sydney. Morgan Kaufmann.

Sato, T. and Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1330–1335, Nagoya, Japan. Morgan Kaufmann.

Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, *4*, 61–76.

Savage, L. J. (1954). *The Foundations of Statistics*. Wiley, New York.

Sayre, K. (1993). Three more flaws in the computational model. Paper presented at the APA (Central Division) Annual Conference, Chicago, Illinois.

Schabes, Y., Abeille, A., and Joshi, A. K. (1988). Parsing strategies with lexicalized grammars: Application to tree adjoining grammars. In Vargha, D. (Ed.), *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Vol. 2, pp. 578–583, Budapest. John von Neumann Society for Computer Science.

Schaeffer, J. (1997). *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, Berlin.

Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum Associates, Potomac, Maryland.

Schank, R. C. and Riesbeck, C. (1981). *Inside Computer Understanding: Five Programs Plus Miniatures*. Lawrence Erlbaum Associates, Potomac, Maryland.

Schapire, R. E. (1999). Theoretical views of boosting and applications. In *Algorithmic Learning Theory: Proceedings of the 10th International Conference (ALT'99)*, pp. 13–25. Springer-Verlag, Berlin.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227.

Schmolze, J. G. and Lipkis, T. A. (1983). Classification in the KL-ONE representation system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pp. 330–332, Karlsruhe, Germany. Morgan Kaufmann.

Schofield, P. D. A. (1967). Complete solution of the eight puzzle. In Dale, E. and Michie, D. (Eds.), *Machine Intelligence 2*, pp. 125–133. Elsevier/North-Holland, Amsterdam, London, New York.

Scholkopf, B. and Smola, A. J. (2002). *Learning with Kernels*. MIT Press, Cambridge, Massachusetts.

Schöning, T. (1999). A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science*, pp. 410–414, New York. IEEE Computer Society Press.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 1039–1046, Milan. Morgan Kaufmann.

Schoppers, M. J. (1989). In defense of reaction plans as caches. *AI Magazine*, *10*(4), 51–60.

Schröder, E. (1877). *Der Operationskreis des Logikkalküls*. B. G. Teubner, Leipzig. .

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*, 1593.

Schütze, H. (1995). *Ambiguity in Language Learning: Computational and Cognitive Models*. Ph.D. thesis, Stanford University. Also published by CSLI Press, 1997.

Schwartz, J. T., Scharir, M., and Hopcroft, J. (1987). *Planning, Geometry and Complexity of Robot Motion*. Ablex Publishing Corporation, Norwood, NJ.

Schwartz, S. P. (Ed.). (1977). *Naming, Necessity, and Natural Kinds*. Cornell University Press, Ithaca, New York.

Scott, D. and Krauss, P. (1966). Assigning probabilities to logical formulas. In Hintikka, J. and Suppes, P. (Eds.), *Aspects of Inductive Logic*. North-Holland, Amsterdam.

Scriven, M. (1953). The mechanical concept of mind. *Mind*, *62*, 230–240.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, UK.

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, *3*, 417–457.

Searle, J. R. (1984). *Minds, Brains and Science*. Harvard University Press, Cambridge, Massachusetts.

Searle, J. R. (1990). Is the brain's mind a computer program?. *Scientific American*, *262*, 26–31.

Searle, J. R. (1992). *The Rediscovery of the Mind*. MIT Press, Cambridge, Massachusetts.

Selman, B., Kautz, H., and Cohen, B. (1996). Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26*, pp. 521–532. American Mathematical Society, Providence, Rhode Island.

Selman, B. and Levesque, H. J. (1993). The complexity of path-based defeasible inheritance. *Artificial Intelligence*, *62*(2), 303–339.

Selman, B., Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 440–446, San Jose. AAAI Press.

Shachter, R. D. (1986). Evaluating influence diagrams. *Operations Research*, *34*, 871–882.

Shachter, R. D. (1998). Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference*, pp. 480–487, Madison, Wisconsin. Morgan Kaufmann.

Shachter, R. D., D'Ambrosio, B., and Del Favero, B. A. (1990). Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 126–131, Boston. MIT Press.

Shachter, R. D. and Kenley, C. R. (1989). Gaussian influence diagrams. *Management Science*, *35*(5), 527–550.

Shachter, R. D. and Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario. Morgan Kaufmann.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey.

Shafer, G. and Pearl, J. (Eds.). (1990). *Readings in Uncertain Reasoning*. Morgan Kaufmann, San Mateo, California.

Shahookar, K. and Mazumder, P. (1991). VLSI cell placement techniques. *Computing Surveys*, *23*(2), 143–220.

Shanahan, M. (1997). *Solving the Frame Problem*. MIT Press, Cambridge, Massachusetts.

Shanahan, M. (1999). The event calculus explained. In Wooldridge, M. J. and Veloso, M. (Eds.), *Artificial Intelligence Today*, pp. 409–430. Springer-Verlag, Berlin.

Shankar, N. (1986). *Proof-Checking Metamathematics*. Ph.D. thesis, Computer Science Department, University of Texas at Austin.

Shannon, C. E. and Weaver, W. (1949). *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, *41*(4), 256–275.

Shapiro, E. (1981). An algorithm that infers theories from facts. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, p. 1064, Vancouver, British Columbia. Morgan Kaufmann.

Shapiro, S. C. (Ed.). (1992). *Encyclopedia of Artificial Intelligence* (second edition). Wiley, New York.

Shapley, S. (1953). Stochastic games. In *Proceedings of the National Academy of Sciences*, Vol. 39, pp. 1095–1100.

Shavlik, J. and Dietterich, T. (Eds.). (1990). *Readings in Machine Learning*. Morgan Kaufmann, San Mateo, California.

Shelley, M. (1818). *Frankenstein: or, the Modern Prometheus*. Pickering and Chatto.

Shenoy, P. P. (1989). A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, *3*(5), 383–411.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *22*(8), 888–905.

Shoham, Y. (1987). Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, *33*(1), 89–104.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–92.

Shoham, Y. (1994). *Artificial Intelligence Techniques in Prolog*. Morgan Kaufmann, San Mateo, California.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*. Elsevier/North-Holland, Amsterdam, London, New York.

Shwe, M. and Cooper, G. (1991). An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network. *Computers and Biomedical Research*, *1991*(5), 453–475.

Siekmann, J. and Wrightson, G. (Eds.). (1983). *Automation of Reasoning*. Springer-Verlag, Berlin.

Sietsma, J. and Dow, R. J. F. (1988). Neural net pruning—why and how. In *IEEE International Conference on Neural Networks*, pp. 325–333, San Diego. IEEE.

Siklossy, L. and Dreussi, J. (1973). An efficient robot planner which generates its own procedures. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pp. 423–430, Stanford, California. IJCAII.

Silverstein, C., Henzinger, M., Marais, H., and Moricz, M. (1998). Analysis of a very large altavista query log. Tech. rep. 1998-014, Digital Systems Research Center.

Simmons, R. and Koenig, S. (1995). Probabilistic robot navigation in partially observable environments. In *Proceedings of IJCAI-95*, pp. 1080–1087, Montreal, Canada. IJCAI, Inc.

Simmons, R. and Slocum, J. (1972). Generating english discourse from semantic networks. *Communications of the ACM*, *15*(10), 891–905.

Simon, H. A. (1947). *Administrative behavior*. Macmillan, New York.

Simon, H. A. (1957). *Models of Man: Social and Rational*. John Wiley, New York.

Simon, H. A. (1963). Experiments with a heuristic compiler. *Journal of the Association for Computing Machinery*, *10*, 493–506.

Simon, H. A. (1981). *The Sciences of the Artificial* (second edition). MIT Press, Cambridge, Massachusetts.

Simon, H. A. (1982). *Models of Bounded Rationality, Volume 1*. The MIT Press, Cambridge, Massachusetts.

Simon, H. A. and Newell, A. (1958). Heuristic problem solving: The next advance in operations research. *Operations Research*, *6*, 1–10.

Simon, H. A. and Newell, A. (1961). Computer simulation of human thinking and problem solving. *Datamation, June/July*, 35–37.

Simon, J. C. and Dubois, O. (1989). Number of solutions to satisfiability instances—applications to knowledge bases. *Int. J. Pattern Recognition and Artificial Intelligence*, *3*, 53–65.

Sirovitch, L. and Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, *2*, 586–591.

Skinner, B. F. (1953). *Science and Human Behavior*. Macmillan, London.

Skolem, T. (1920). Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über die dichte Mengen. *Videnskapsselskapets skrifter, I. Matematisk-naturvidenskabelig klasse*, *4*.

Skolem, T. (1928). Über die mathematische Logik. *Norsk matematisk tidsskrift*, *10*, 125–142.

Slagle, J. R. (1963a). A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the Association for Computing Machinery*, *10*(4).

Slagle, J. R. (1963b). Game trees, m & n minimaxing, and the m & n alpha–beta procedure. Artificial intelligence group report 3, University of California, Lawrence Radiation Laboratory, Livermore, California.

Slagle, J. R. and Dixon, J. K. (1969). Experiments with some programs that search game trees. *Journal of the Association for Computing Machinery*, *16*(2), 189–207.

Slate, D. J. and Atkin, L. R. (1977). CHESS 4.5—Northwestern University chess program. In Frey, P. W. (Ed.), *Chess Skill in Man and Machine*, pp. 82–118. Springer-Verlag, Berlin.

Slater, E. (1950). Statistics for the chess computer and the factor of mobility. In *Symposium on Information Theory*, pp. 150–152, London. Ministry of Supply.

Sleator, D. and Temperley, D. (1993). Parsing English with a link grammar. In *Third Annual Workshop on Parsing technologies*.

Sloman, A. (1978). *The Computer Revolution in Philosophy*. Harvester Press, Hassocks, Sussex, UK.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, *21*, 1071–1088.

Smith, D. E., Genesereth, M. R., and Ginsberg, M. L. (1986). Controlling recursive inference. *Artificial Intelligence*, *30*(3), 343–389.

Smith, D. R. (1990). KIDS: a semiautomatic program development system. *IEEE Transactions on Software Engineering*, *16*(9), 1024–1043.

Smith, D. R. (1996). Machine support for software development. In *Proceedings of the 18th International Conference on Software Engineering*, pp. 167–168, Berlin. IEEE Computer Society Press.

Smith, D. E. and Weld, D. S. (1998). Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, p. ???, Madison, Wisconsin. AAAI Press.

Smith, J. Q. (1988). *Decision Analysis*. Chapman and Hall, London.

Smith, J. M. and Szathmáry, E. (1999). *The Origins of Life: From the Birth of Life to the Origin of Language*. Oxford University Press, Oxford, UK.

Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, *5*(4), 56–68.

Smith, S. J. J., Nau, D. S., and Throop, T. A. (1998). Success in spades: Using ai planning techniques to win the world championship of computer bridge. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 1079–1086, Madison, Wisconsin. AAAI Press.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, *2*, 1–74.

Smyth, P., Heckerman, D., and Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, *9*(2), 227–269.

Soderland, S. and Weld, D. S. (1991). Evaluating nonlinear planning. Technical report TR-91-02-03, University of Washington Department of Computer Science and Engineering, Seattle, Washington.

Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control*, *7*, 1–22, 224–254.

Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. Ph.D. thesis, Stanford University, Stanford, California.

Sosic, R. and Gu, J. (1994). Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, *6*(5), 661–668.

Sowa, J. (1999). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Blackwell, Oxford, UK.

Spiegelhalter, D. J. (1986). Probabilistic reasoning in predictive expert systems. In Kanal, L. N. and Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence*, pp. 47–67. Elsevier/North-Holland, Amsterdam, London, New York.

Spiegelhalter, D. J., Dawid, P., Lauritzen, S., and Cowell, R. (1993). Bayesian analysis in expert systems. *Statistical Science*, *8*, 219–282.

Spielberg, S. (2001). AI. movie.

Spirtes, P., Glymour, C., and Scheines, R. (1993). *Causation, prediction, and search*. Springer-Verlag, Berlin.

Springsteen, B. (1992). 57 channels (and nothin' on). In *Human Touch*. Sony.

Srinivasan, A., Muggleton, S. H., King, R. D., and Sternberg, M. J. E. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In Wrobel, S. (Ed.), *Proceedings of the 4th International Workshop on Inductive Logic Programming*, Vol. 237, pp. 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH.

Srivas, M. and Bickford, M. (1990). Formal verification of a pipelined microprocessor. *IEEE Software*, *7*(5), 52–64.

Stallman, R. M. and Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, *9*(2), 135–196.

Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *Communications of the Association for Computing Machinery*, *29*(12), 1213–1228.

Stefik, M. (1995). *Introduction to Knowledge Systems*. Morgan Kaufmann, San Mateo, California.

Stein, L. A. (2002). *Interactive Programming in Java (pre-publication draft)*. Morgan Kaufmann, San Mateo, California.

Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, pp. 109–110. ACM Press.

Stevens, K. A. (1981). The information content of texture gradients. *Biological Cybernetics*, *42*, 95–105.

Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, *1*(4), 333–355.

Stickel, M. E. (1988). A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, *4*, 353–380.

Stiller, L. B. (1992). KQNKRR. *ICCA Journal*, *15*(1), 16–18.

Stillings, N. A., Weisler, S., Feinstein, M. H., Garfield, J. L., and Rissland, E. L. (1995). *Cognitive Science: An Introduction* (second edition). MIT Press, Cambridge, Massachusetts.

Stockman, G. (1979). A minimax algorithm better than alpha–beta?. *Artificial Intelligence*, *12*(2), 179–196.

Stolcke, A. and Omohundro, S. (1994). Inducing probabilistic grammars by Bayesian model merging.. In *Proceedings of the Second International Colloquium on Grammatical Inference and Applications (ICGI-94)*, pp. 106–118, Alicante, Spain. Springer-Verlag.

Stone, P. (2000). *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*. MIT Press, Cambridge, Massachusetts.

Strachey, C. (1952). Logical or non-mathematical programmes. In *Proceedings of the Association for Computing Machinery (ACM)*, pp. 46–49, Toronto, Canada.

Subramanian, D. (1993). Artificial intelligence and conceptual design. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 800–809, Chambery, France. Morgan Kaufmann.

Subramanian, D. and Feldman, R. (1990). The utility of EBL in recursive domain theories. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Vol. 2, pp. 942–949, Boston. MIT Press.

Subramanian, D. and Wang, E. (1994). Constraint-based kinematic synthesis. In *Proceedings of the International Conference on Qualitative Reasoning*, pp. 228–239. AAAI Press.

Sugihara, K. (1984). A necessary and sufficient condition for a picture to represent a polyhedral scene. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *6*(5), 578–586.

Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. Elsevier/North-Holland, Amsterdam, London, New York.

Sussman, G. J. and Winograd, T. (1970). MICRO-PLANNER Reference Manual. Ai memo 203, MIT AI Lab, Cambridge, Massachusetts.

Sutherland, I. (1963). Sketchpad: A man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference*, pp. 329–346. IFIPS.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K.-R. (Eds.), *Advances in Neural Information Processing Systems 12*, pp. 1057–1063. MIT Press, Cambridge, Massachusetts.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning: Proceedings of the Seventh International Conference*, pp. 216–224, Austin, Texas. Morgan Kaufmann.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts.

Swade, D. D. (1993). Redeeming Charles Babbage's mechanical computer. *Scientific American*, *268*(2), 86–91.

Swerling, P. (1959). First order error propagation in a stagewise smoothing procedure for satellite observations. *Journal of Astronautical Sciences*, *6*, 46–52.

Swift, T. and Warren, D. S. (1994). Analysis of SLG-WAM evaluation of definite programs. In *Logic Programming. Proceedings of the 1994 International Symposium*, pp. 219–235, Ithaca, NY. MIT Press.

Syrjänen, T. (2000). Lparse 1.0 user's manual. http://saturn.tcs.hut.fi/Software/smodels.

Tadepalli, P. (1993). Learning from queries and examples with tree-structured bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 322–329, Amherst, Massachusetts. Morgan Kaufmann.

Tait, P. G. (1880). Note on the theory of the "15 puzzle". *Proceedings of the Royal Society of Edinburgh*, *10*, 664–665.

Tamaki, H. and Sato, T. (1986). OLD resolution with tabulation. In *Third International Conference on Logic Programming*, pp. 84–98, London. Springer-Verlag.

Tambe, M., Newell, A., and Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, *5*, 299–348.

Tarjan, R. E. (1983). *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM (Society for Industrial and Applied Mathematics, Philadelphia.

Tarski, A. (1935). Die Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, *1*, 261–405.

Tarski, A. (1956). *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Oxford University Press, Oxford, UK.

Tash, J. K. and Russell, S. J. (1994). Control strategies for a stochastic planner. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1079–1085, Seattle. AAAI Press.

Tate, A. (1975a). Interacting goals and their use. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pp. 215–218, Tbilisi, Georgia. IJCAII.

Tate, A. (1975b). *Using Goal Structure to Direct Search in a Problem Solver*. Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland.

Tate, A. (1977). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pp. 888–893, Cambridge, Massachusetts. IJCAII.

Tate, A. and Whiter, A. M. (1984). Planning with multiple resource constraints and an application to a naval planning problem. In *Proceedings of the First Conference on AI Applications*, pp. 410–416, Denver, Colorado.

Tatman, J. A. and Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, *20*(2), 365–379.

Tesauro, G. (1989). Neurogammon wins computer olympiad. *Neural Computation*, *1*(3), 321–323.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*(3–4), 257–277.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the Association for Computing Machinery*, *38*(3), 58–68.

Tesauro, G. and Sejnowski, T. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence*, *39*(3), 357–390.

Thagard, P. (1996). *Mind: Introduction to Cognitive Science*. MIT Press, Cambridge, Massachusetts.

Thaler, R. (1992). *The Winner's Curse: Paradoxes and Anomalies of Economic Life*. Princeton University Press, Princeton, New Jersey.

Thielscher, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, *111*(1–2), 277–299.

Thomason, R. H. (Ed.). (1974). *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, Connecticut.

Thompson, D. W. (1917). *On Growth and Form*. Cambridge University Press, Cambridge, UK.

Thrun, S. (2000). Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA. IEEE.

Thrun, S. (2002). Robotic mapping: A survey. In Lakemeyer, G. and Nebel, B. (Eds.), *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann. to appear.

Titterington, D. M., Smith, A. F. M., and Makov, U. E. (1985). *Statistical analysis of finite mixture distributions*. Wiley, New York.

Toffler, A. (1970). *Future Shock*. Bantam.

Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, *9*, 137–154.

Touretzky, D. S. (1986). *The Mathematics of Inheritance Systems*. Pitman and Morgan Kaufmann, London and San Mateo, California.

Trucco, E. and Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, Upper Saddle River, New Jersey.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, New York.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674–690.

Tumer, K. and Wolpert, D. (2000). Collective intelligence and braess' paradox. In *Proceedings of the AAAI/IAAI*, pp. 104–109.

Turcotte, M., Muggleton, S. H., and Sternberg, M. J. E. (2001). Automated discovery of structural signatures of protein fold and function. *Journal of Molecular Biology*, *306*, 591–605.

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series*, *42*, 230–265.

Turing, A. (1948). Intelligent machinery. Tech. rep., National Physical Laboratory. reprinted in (Ince, 1992).

Turing, A. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.

Turing, A., Strachey, C., Bates, M. A., and Bowden, B. V. (1953). Digital computers applied to games. In Bowden, B. V. (Ed.), *Faster than Thought*, pp. 286–310. Pitman, London.

Turtle, H. R. and Croft, W. B. (1992). A comparison of text retrieval models. *The Computer Journal*, *35*(1), 279–289.

Tversky, A. and Kahneman, D. (1982). Causal schemata in judgements under uncertainty. In Kahneman, D., Slovic, P., and Tversky, A. (Eds.), *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge, UK.

Ullman, J. D. (1985). Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, *10*(3), 289–321.

Ullman, J. D. (1989). *Principles of Database and Knowledge-Base Bystems*. Computer Science Press, Rockville, Maryland.

Ullman, S. (1979). *The Interpretation of Visual Motion*. MIT Press, Cambridge, Massachusetts.

Vaessens, R. J. M., Aarts, E. H. I., and Lenstra, J. K. (1996). Job shop scheduling by local search. *INFORMS J. on Computing*, *8*, 302–117.

Valiant, L. (1984). A theory of the learnable. *Communications of the Association for Computing Machinery*, *27*, 1134–1142.

van Benthem, J. (1983). *The Logic of Time*. D. Reidel, Dordrecht, Netherlands.

Van Emden, M. H. and Kowalski, R. (1976). The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery*, *23*(4), 733–742.

van Harmelen, F. and Bundy, A. (1988). Explanation-based generalisation = partial evaluation. *Artificial Intelligence*, *36*(3), 401–412.

van Heijenoort, J. (Ed.). (1967). *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, Massachusetts.

Van Hentenryck, P., Saraswat, V., and Deville, Y. (1998). Design, implementation, and evaluation of the constraint language cc(fd). *Journal of Logic Programming*, *37*(1–3), 139–164.

van Nunen, J. A. E. E. (1976). A set of successive approximation methods for discounted Markovian decision problems. *Zeitschrift fur Operations Research, Serie A*, *20*(5), 203–208.

van Roy, B. (1998). *Learning and value function approximation in complex decision processes*. Ph.D. thesis, Laboratory for Information and Decision Systems, MIT, Cambridge, Massachusetts.

Van Roy, P. L. (1990). Can logic programming execute as fast as imperative programming?. Report UCB/CSD 90/600, Computer Science Division, University of California, Berkeley, California.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley, New York.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, *16*, 264–280.

Varian, H. R. (1995). Economic mechanism design for computerized agents. In *USENIX Workshop on Electronic Commerce*, pp. 13–21.

Veloso, M. and Carbonell, J. G. (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, *10*, 249–278.

Vere, S. A. (1983). Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *5*, 246–267.

Vinge, V. (1993). The coming technological singularity: How to survive in the post-human era. In *VISION-21 Symposium*. NASA Lewis Research Center and the Ohio Aerospace Institute.

Viola, P. and Jones, M. (2002). Robust real-time object detection. *International Journal of Computer Vision*, in press.

von Mises, R. (1928). *Wahrscheinlichkeit, Statistik und Wahrheit*. J. Springer, Berlin.

von Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, *100*(295–320).

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior* (first edition). Princeton University Press, Princeton, New Jersey.

von Winterfeldt, D. and Edwards, W. (1986). *Decision Analysis and Behavioral Research*. Cambridge University Press, Cambridge, UK.

Voorhees, E. M. (1993). Using WordNet to disambiguate word senses for text retrieval. In *Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 171–80, Pittsburgh. Association for Computing Machinery.

Vossen, T., Ball, M., Lotem, A., and Nau, D. S. (2001). Applying integer programming to ai planning. *Knowledge Engineering Review*, *16*, 85–100.

Waibel, A. and Lee, K.-F. (1990). *Readings in Speech Recognition*. Morgan Kaufmann, San Mateo, California.

Waldinger, R. (1975). Achieving several goals simultaneously. In Elcock, E. W. and Michie, D. (Eds.), *Machine Intelligence 8*, pp. 94–138. Ellis Horwood, Chichester, England.

Waltz, D. (1975). Understanding line drawings of scenes with shadows. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*. McGraw-Hill, New York.

Wanner, E. (1974). *On remembering, forgetting and understanding sentences.* Mouton, The Hague and Paris.

Warren, D. H. D. (1974). WARPLAN: A System for Generating Plans. Department of Computational Logic Memo 76, University of Edinburgh, Edinburgh, Scotland.

Warren, D. H. D. (1976). Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, pp. 344–354.

Warren, D. H. D. (1983). An abstract Prolog instruction set. Technical note 309, SRI International, Menlo Park, California.

Warren, D. H. D., Pereira, L. M., and Pereira, F. (1977). PROLOG: The language and its implementation compared with LISP. *SIGPLAN Notices*, *12*(8), 109–115.

Watkins, C. J. (1989). *Models of Delayed Reinforcement Learning*. Ph.D. thesis, Psychology Department, Cambridge University, Cambridge, UK.

Watson, J. D. and Crick, F. H. C. (1953). A structure for deoxyribose nucleic acid. *Nature*, *171*, 737.

Webber, B. L. (1983). So what can we talk about now?. In Brady, M. and Berwick, R. (Eds.), *Computational Models of Discourse*. MIT Press, Cambridge, Massachusetts.

Webber, B. L. (1988). Tense as discourse anaphora. *Computational Linguistics*, *14*(2), 61–73.

Webber, B. L. and Nilsson, N. J. (Eds.). (1981). *Readings in Artificial Intelligence*. Morgan Kaufmann, San Mateo, California.

Weidenbach, C. (2001). SPASS: Combining superposition, sorts and splitting. In Robinson, A. and Voronkov, A. (Eds.), *Handbook of Automated Reasoning*. mit, mit-ad.

Weiss, G. (1999). *Multiagent systems*. MIT Press, Cambridge, Massachusetts.

Weiss, S. and Kulikowski, C. A. (1991). *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann, San Mateo, California.

Weizenbaum, J. (1976). *Computer Power and Human Reason*. W. H. Freeman, New York.

Weld, D. S. (1994). An introduction to least commitment planning. *AI Magazine*, *15*(4), 27–61.

Weld, D. S. (1999). Recent advances in ai planning. *AI Magazine*, *20*(2), 93–122.

Weld, D. S., Anderson, C. R., and Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 897–904, Madison, Wisconsin. AAAI Press.

Weld, D. S. and de Kleer, J. (1990). *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, California.

Weld, D. S. and Etzioni, O. (1994). The first law of robotics: A call to arms. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle. AAAI Press.

Wellman, M. P. (1985). Reasoning about preference models. Technical report MIT/LCS/TR-340, Laboratory for Computer Science, MIT, Cambridge, Massachusetts.

Wellman, M. P. (1988). *Formulation of Tradeoffs in Planning under Uncertainty*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Wellman, M. P. (1990a). Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, *44*(3), 257–303.

Wellman, M. P. (1990b). The STRIPS assumption for planning under uncertainty. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 198–203, Boston. MIT Press.

Wellman, M. P. (1995). The economic approach to artificial intelligence. *ACM Computing Surveys*, *27*(3), 360–362.

Wellman, M. P., Breese, J. S., and Goldman, R. (1992). From knowledge bases to decision models. *Knowledge Engineering Review*, *7*(1), 35–53.

Wellman, M. P. and Doyle, J. (1992). Modular utility representation for decision-theoretic planning. In *Proceedings, First International Conference on AI Planning Systems*, pp. 236–242, College Park, Maryland. Morgan Kaufmann.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University, Cambridge, Massachusetts.

Werbos, P. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, *22*, 25–38.

Wesley, M. A. and Lozano-Perez, T. (1979). An algorithm for planning collision-free paths among polyhedral objects. *Communications of the ACM*, *22*(10), 560–570.

Wheatstone, C. (1838). On some remarkable, and hitherto unresolved, phenomena of binocular vision. *Philosophical Transactions of the Royal Society of London*, *2*, 371–394.

Whitehead, A. N. (1911). *An Introduction to Mathematics*. Williams and Northgate, London.

Whitehead, A. N. and Russell, B. (1910). *Principia Mathematica*. Cambridge University Press, Cambridge, UK.

Whorf, B. (1956). *Language, Thought, and Reality*. MIT Press, Cambridge, Massachusetts.

Widrow, B. (1962). Generalization and information storage in networks of adaline "neurons". In Yovits, M. C., Jacobi, G. T., and Goldstein, G. D. (Eds.), *Self-Organizing Systems 1962*, pp. 435–461, Chicago, Illinois. Spartan.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pp. 96–104, New York.

Wiener, N. (1942). The extrapolation, interpolation, and smoothing of stationary time series. Osrd 370, Report to the Services 19, Research Project DIC-6037, MIT, Cambridge, Massachusetts.

Wiener, N. (1948). *Cybernetics*. Wiley, New York.

Wilensky, R. (1978). *Understanding goal-based stories*. Ph.D. thesis, Yale University, New Haven, Connecticut.

Wilensky, R. (1983). *Planning and Understanding*. Addison-Wesley, Reading, Massachusetts.

Wilkins, D. E. (1980). Using patterns and plans in chess. *Artificial Intelligence*, *14*(2), 165–203.

Wilkins, D. E. (1988). *Practical Planning: Extending the AI Planning Paradigm*. Morgan Kaufmann, San Mateo, California.

Wilkins, D. E. (1990). Can AI planners solve practical problems?. *Computational Intelligence*, *6*(4), 232–246.

Wilkins, D. E., Myers, K. L., Lowrance, J. D., and Wesley, L. P. (1995). Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, *7*(1), 197–227.

Wilks, Y. (1975). An intelligent analyzer and understander of English. *Communications of the ACM*, *18*(5), 264–274.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Williams, R. J. and Baird, L. C. I. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep. NU-CCS-93-14, College of Computer Science, Northeastern University, Boston.

Wilson, R. A. and Keil, F. C. (Eds.). (1999). *The MIT Encyclopedia of the Cognitive Sciences*. MIT Press, Cambridge, Massachusetts.

Winograd, S. and Cowan, J. D. (1963). *Reliable Computation in the Presence of Noise*. MIT Press, Cambridge, Massachusetts.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, *3*(1), 1–191.

Winston, P. H. (1970). Learning structural descriptions from examples. Technical report MAC-TR-76, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Winston, P. H. (1992). *Artificial Intelligence* (Third edition). Addison-Wesley, Reading, Massachusetts.

Wirth, R. and O'Rorke, P. (1991). Constraints on predicate invention. In *Machine Learning: Proceedings of the Eighth International Workshop (ML-91)*, pp. 457–461, Evanston, Illinois. Morgan Kaufmann.

Witten, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, *37*(4), 1085–1094.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (second edition). Morgan Kaufmann, San Mateo, California.

Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus* (second edition). Routledge and Kegan Paul, London. Reprinted 1971, edited by D. F. Pears and B. F. McGuinness. This edition of the English translation also contains Wittgenstein's original German text on facing pages, as well as Bertrand Russell's introduction to the 1922 edition.

Wittgenstein, L. (1953). *Philosophical Investigations*. Macmillan, London.

Wojciechowski, W. S. and Wojcik, A. S. (1983). Automated design of multiple-valued logic circuits by automated theorem proving techniques. *IEEE Transactions on Computers*, *C-32*(9), 785–798.

Wojcik, A. S. (1983). Formal design verification of digital systems. In *ACM IEEE 20th Design Automation Conference Proceedings*, pp. 228–234, Miami Beach, Florida. IEEE.

Wood, M. K. and Dantzig, G. B. (1949). Programming of interdependent activities. i. general discussion. *Econometrica*, *17*, 193–199.

Woods, W. A. (1973). Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings*, Vol. 42, pp. 441–450.

Woods, W. A. (1975). What's in a link? Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M. (Eds.), *Representation and Understanding: Studies in Cognitive Science*, pp. 35–82. Academic Press, New York.

Woods, W. A. (1978). Semantics and quantification in natural language question answering. In *Advances in Computers*. Academic Press.

Wooldridge, M. and Rao, A. (Eds.). (1999). *Foundations of rational agency*. Kluwer, Dordrecht, Netherlands.

Wos, L., Carson, D., and Robinson, G. (1964). The unit preference strategy in theorem proving. In *Proceedings of the Fall Joint Computer Conference*, pp. 615–621.

Wos, L., Carson, D., and Robinson, G. (1965). Efficiency and completeness of the set-of-support strategy in theorem proving. *Journal of the Association for Computing Machinery*, *12*, 536–541.

Wos, L., Overbeek, R., Lusk, E., and Boyle, J. (1992). *Automated Reasoning: Introduction and Applications* (second edition). McGraw-Hill, New York.

Wos, L. and Robinson, G. (1968). Paramodulation and set of support. In *Proceedings of the IRIA Symposium on Automatic Demonstration*, pp. 276–310. Springer-Verlag.

Wos, L., Robinson, G., Carson, D., and Shalla, L. (1967). The concept of demodulation in theorem proving. *Journal of the Association for Computing Machinery*, *14*, 698–704.

Wos, L. and Winker, S. (1983). Open questions solved with the assistance of AURA. In Bledsoe, W. W. and Loveland, D. (Eds.), *Automated Theorem Proving: After 25 Years: Proceedings of the Special Session of the 89th Annual Meeting of the American Mathematical Society*, pp. 71–88, Denver, Colorado. American Mathematical Society.

Wright, S. (1921). Correlation and causation. *Journal of Agricultural Research*, *20*, 557–585.

Wright, S. (1931). Evolution in Mendelian populations. *Genetics*, *16*, 97–159.

Wright, S. (1934). The method of path coefficients. *Annals of Mathematical Statistics*, *5*, 161–215.

Wu, D. (1993). Estimating probability distributions over hypotheses with variable unification. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 790–795, Chambery, France. Morgan Kaufmann.

Wygant, R. M. (1989). CLIPS— a powerful development and delivery expert system tool. *Computers and Industrial Engineering*, *17*, 546–549.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of the Thirty Ninth Annual Conference of the Association for Computational Linguistics*, pp. 228–235.

Yang, Q. (1990). Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, *6*, 12–24.

Yang, Q. (1997). *Intelligent planning: A decomposition and abstraction based approach*. Springer-Verlag, Berlin.

Yedidia, J., Freeman, W., and Weiss, Y. (2001). Generalized belief propagation. In Leen, T. K., Dietterich, T., and Tresp, V. (Eds.), *Advances in Neural Information Processing Systems 13*. MIT Press, Cambridge, Massachusetts.

Yip, K. M.-K. (1991). *KAM: A System for Intelligently Guiding Numerical Experimentation by Computer*. MIT Press, Cambridge, Massachusetts.

Yngve, V. (1955). A model and an hypothesis for language structure. In Locke, W. N. and Booth, A. D. (Eds.), *Machine Translation of Languages*, pp. 208–226. MIT Press, Cambridge, Massachusetts.

Yob, G. (1975). Hunt the wumpus!. *Creative Computing*, *Sep/Oct*.

Yoshikawa, T. (1990). *Foundations of Robotics: Analysis and Control*. MIT Press, Cambridge, Massachusetts.

Young, R. M., Pollack, M. E., and Moore, J. D. (1994). Decomposition and causality in partial order planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pp. 188–193, Chicago.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, *10*(2), 189–208.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.

Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, *1*, 3–28.

Zaritskii, V. S., Svetnik, V. B., and Shimelevich, L. I. (1975). Monte-Carlo technique in problems of optimal information processing. *Automation and Remote Control*, *36*, 2015–22.

Zelle, J. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 1050–1055.

Zermelo, E. (1913). Uber Eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proceedings of the Fifth International Congress of Mathematicians*, Vol. 2, pp. 501–504.

Zermelo, E. (1976). An application of set theory to the theory of chess-playing. *Firbush News*, *6*, 37–42. English translation of (Zermelo 1913).

Zhang, N. L. and Poole, D. (1994). A simple approach to bayesian network computations. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, pp. 171–178, Banff, Alberta. Morgan Kaufmann.

Zhang, N. L. and Poole, D. (1996). Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, *5*, 301–328.

Zhou, R. and Hansen, E. (2002). Memory-bounded A* graph search. In *Proceedings of the 15th International Flairs Conference*.

Zhu, D. J. and Latombe, J.-C. (1991). New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, *7*(1), 9–20.

Zilberstein, S. and Russell, S. J. (1996). Optimal composition of real-time systems. *Artificial Intelligence*, *83*, 181–213.

Zimmermann, H.-J. (Ed.). (1999). *Practical applications of fuzzy technologies*. Kluwer, Dordrecht, Netherlands.

Zimmermann, H.-J. (2001). *Fuzzy Set Theory—And Its Applications* (Fourth edition). Kluwer, Dordrecht, Netherlands.

Zobrist, A. L. (1970). *Feature Extraction and Representation for Pattern Recognition and the Game of Go*. Ph.D. thesis, University of Wisconsin.

Zuse, K. (1945). The Plankalkül. Report 175, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, Germany.

Zweig, G. and Russell, S. J. (1998). Speech recognition with dynamic Bayesian networks. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 173–180, Madison, Wisconsin. AAAI Press.

Índice alfabético

$\prec$ (antes), 443
• (en el análisis), 908
$A \xrightarrow{p} B$ (logra), 444
λ-expresión, 280
$\Rightarrow$ (implica), 229
$\wedge$ (y), 229
$\Leftrightarrow$ (si y sólo si), 230
$\neg$ (no), 229
$\vee$ (o), 229
$\mapsto$ (regla incierta), 597
$\succ$(determinación), 791
$\Box p$ (siempre), 414
$\Diamond p$ (eventualmente), 414
$=$ (derivación), 225
$\vdash$ (se deriva), 227
$\exists$ (existe), 283
$\forall$ (para todos), 282
$\sim$ *(indiferente),* 666
$\succ$(preferido, 666
$u_\top$ (mejor precio), 673
$u_\perp$ (peor catástrofe), 673
χ^2 (chi al cuadrado), 775
ϵ-bola, **762**

AAAI (American Association for AI, Asociación Americana para la IA), 35
Aarts, E. H. L, 1173
Aarup, M., 519, 1121
Abbott, L. E, 863, 1132
Abeille, A., 1167
Abelson, R. P., 28, 938, 1167
ABO (Optimalidad Asintótica Limitada), 1105
Abramson, B., 99, 1121
abrir-el-código, 331
ABSOLVER, 122
abstracción, **72**
ABSTRIPS, 519
AC-3, **165-176**
AC-4, 176
accesible, 47
ACCIÓN, 78, 81
acción, 38, 97
  aplicable, **430**
  condicional, 520
  conjunta, **514**
  de descomposición, **481**, 482-484
  de descomposición (recursiva), 488
  de ejecución, 502, **503**, 504, 505
  esquema, **430**
  primitiva, 481
  racional, 8, 33
  sensorial, **491**, **501**, 520, 682
  tabla acción-utilidad, **680**
ACCIONES, 139, 142
acciones
  persistentes, **451**
  sensoriales, 491, **501**
    activo, **501**, 502
    automático, **501**
Acero, A., 1144
Acharya, A., 1128
aciclicidad, **259**
acidez, 40
Ackley, D. H., 149, 1121
acortamiento delantero, 998
ACT, 325
ACT*, 806
actitudes de proposición, 387
activación, 839
acto del habla, **898**, 898, 936
actuación neumática, **1029**
actuador, **38**, 45
Actualización de Bellman, **705**
ACTUALIZACIÓN-ESPACIO-VERSIÓN, **779**, 781
ACTUALIZAR-ESTADO, 56, 69, 903
actuar racionalmente, 5
acuerdo (en una sentencia), **914**
acumulación, 579
acústica (de la inferencia), **227**
ADABOOST, 759, **759**, 768
adaline, 23
Adams, J., 386
Adelson-Velsky, G. M., 210, 1121
adjunto, **918**
ADL (Lenguaje de Descripción de Acciones), **432**, 466
admisible, 439
Adorf, H.-M., 519, 1146
ADP (Programación Dinámica Adaptativa) Dynamic Programming), 872
adquisición de conocimiento, 22, 27, **296**
  analógico, 304
  anterior, 43
  base de, **219**, 261, 304
  diagnóstico, 548
  efecto, **391**
  fuente, 658
  ingeniería del, **295**, 295-302, 364, 565
  lenguaje, **219**, 261, 271
  mapa, *véase* red Bayesiana basado en modelos, 548
  nivel, **220**, 263
  para sistemas téoricos de decisión, 688
  precondición, **391**

proposición, **258**
representación, 3, 19, 22, 28, 271-276, 363-425
sistema basado en, 26-28, 881
y acción, 8, 390
aeropuerto, conducción hacia, 527
afirmación de caso (en planes de condición), 495, 496
Agencia Espacial Europea, 519
agente, 5, 37, 62
activo, 876
ADP pasivo, 895
arquitectura de, 31, 1102-1103
aspiradora, 40, 65
autónomo, 220
basado en
circuitos, 256-260
conocimiento, 15, 217, 220, 271, 1099
modelos, 55, 56, **56**
utilidad, 58, 61, 716
componentes, 1099-1102
conducción de taxis, 61, 1103
de aprendizaje, 60-62, 64
pasivo, 895
función, 38, **39**, 699
ingenuo, 902
inteligente, 33, 1093, 1099
lógico, 219, 253-261, 304
orientado a metas, 57, 63
pájaro, **516**, 522
pasivo, 869
planificación continua, 507
programa, **39**, **51**, **52**, **62**
racional, 4-5, 37, 42, 63, 690, 1099
limitado, 389
reflejo, 53-56, 62
remoto, 32, 64, 352, 464
replanificación, 518
resolvente de problemas, 67, 67-72
situado, 1081
software, **46**
teórico basado en decisión, 531, 664, 716-718
voraz, 876
wumpus, 221-224, 292, 527, 980
AGENTE,
ADP-PASIVO, **873**, 771
APRENDIZAJE-Q, **881**
BC, **219**
BPP-ONLINE, 141, **142**, 145, 129
DIRIGIDO MEDIANTE-TABLA, 52
DT, **532**
POP-CONTINUO, 512, 521
REACTIVO-CON-ESTADO, **56**
REACTIVO-SIMPLE, **53**, 54
REPLANIFICACIÓN, 503
TD-PASIVO, **874**
WUMPUS-LP, 254, **254**, 260, 269
Agmon, S., 861, 1121
Agre, P. E., 521, 1121
agregación, **480**
agrupación de documentos, **957**
Agrupamiento, 602, 826-829
(en redes Bayesianas), **581**, 581
aguja en un pajar, 227
Aho, A. V., 938, 1115, 1121
AI, *véase* Inteligencia Artificial
AI en tiempo real, *véase* inteligencia artificial, en tiempo real
AISB (Society for Artificial Intelligence and Simulation of Behaviour, Sociedad para la Inteligencia Artificial y Simulación del comportamiento), 35
Ait-Kaci, 357, 1121
Aizerman, M., 863, 1121
ajedrez, 16, 24, 49, 100, 182, 193-194, 202-203, 208
autómata, 208
alarma antirobo, 562-564
albedo, 985
Alberti, L. B., 1015
Albus, J. S., 892, 1121
alcanza, **444**
Aldiss, B., 1095
Aldous, D., 148, 1121
aleatoria, **533**, 566
continua, 571, 603
aleatorización, 38, 55
Alejandría, 17
alfa-beta, *véase* búsqueda, poda alfa-beta alfa-beta, **188**, 214
alfabeto fonético, 646
álgebra de Robbins, **351**
algoritmo, **9**
de cualquier cualquier momento, **1103**
de Monte Carlo, **582**
dentro-fuera, **951**
genético, 25, **131**, 131-134, 149-150
Húngaro, 1010
Metrópolis, 149, 603
minimax, 185, 495
óptimamente eficiente, **114**
simplex, 149
ALGORITMO-GENÉTICO, **134**
Alhazen, 1015
ALICE, 1077
ALINEACIÓN, 1011, **1012**
ALisp, **1061**
al-Khowarazmi, 9
Allais, M., 672, 1121
Allen, B., 1137
Allen, J. F., 415, 469, 941, 1121
Allen, W., 652
alma, 1096
almacenan el caso, **956**
ALMACENAR, 316, 359
Almuallim, H., 807, 1121
Almulla, M., 1161
ALPAC., 975, 1121
Alperin Resnick, L., 1126
Alshawi, H., 938, 1121
ALTA, 796
altavoz sustancial, **897**
Alterman, R., 520, 1121
altruismo, 531
ALVINN, 32
AM, 808
(árbol de recubrimiento mínimo), 151, 152
Amarel, S., 99, 102, 412, 1121
ambigüedad, 273, 395, 900, 927-930
sintáctica, 938
Ambros-Ingerson, J., 521, 1121
Amit, D., 861, 1121
amor, 1077
amplificación, **914**
amplio contenido, **1085**
análisis, **900**, 905-913
ascendente, **906**
asintótico, 1109-1110
de abajo a arriba, **906**
de algoritmos, **1109**
de arriba a descendente, **906**
de complejidad, 83, **1110**
de decisión, **686**
de descendente, **906**
de sensibilidad, 689
grafo, 908
analista decisión, **687**
analogía derivacional, 807
ANALOGY, 23, 36
Anantharaman, T. S., 1144
Anbulagan, 265, 1152
ancho de hiperárbol, 580
anchura de árbol, **175**, 178

Andersen, S. K., 602, 1122
Anderson, A. R., 1096, 1122
Anderson, C. R., 1174
Anderson, C. W., 1123
Anderson, J. A., 35, 861, 1122, 1143
Anderson, J. R., 16, 325, 604, 806, 1122, 1162
Andre, D., 894, 1061, 1122, 1150
anomalía de Sussman , 467, **471**
Anshelevich, V. A., 212, 1122
antecedente, 229
anterior uniforme, 815
Anthony, M., 769, 862, 1122
Antigua y buena IA (good old-fashioned AI), 1079, 1080, 1097
AÑADIR-ARACO, 909, 910
añadir lista (en STRIPS), **431**
Appel, K., 176, 1122
Appelt, D., 1143
aprendizaje, **43**, **63**, 220, 739, 1077, 1081
  acción-valor,868
  activo, **869**
  acumulativo, 797, 805
  automático, 3, 6
  basado en explicación, 784, 786-791, 805
  basado en relevancia. **785**, 791-795, 805
  Bayesiano, **812**, 812-812, 858
  computacional, 760
  conjunto, 756-760
  de diferencias temporales 872-876, 891
  damas, 21
  estimación de eficiencia, 660-661
  incremental, 779
  micromundos, 23
  por refuerzo, **740**, 869-896, 1080
    activo, 876-882
    generalización en, 882-887
  problema del carro y el pértiga, 886
  programación lógica inductiva (ILP), 807
  red Bayesiana, 823-824
  supervisado, **740**, 1081
  teoría computacional, 761
APRENDIZAJE-
  ESPACIO-VERSIÓN, **779**, 782
  PERCEPTRON, 845
  POR RETROPROPAGACIÓN, **849**
a priori optimista, 878
Apt, K. R., 177, 1122
Apté, C., 974, *1122*
apuesta de Pascal, 555, 690
árbol
  de decisión, 692, 744
    expresividad, 745
    poda, **754**
  empaquetado, 943
  genealógico, 795
  mínimo de recubrimiento, 147, 151
  sintáctico, **900**
árboles de uniones, **580**
Arbuckle, T., 1124
arco
  consistente, **164**
  de persistencia, **639**
área de trabajo, **1040**
Arentoft, M. M., 1121
Ares, M., *1127*
ARGS, 316
argumento desde la
  discapacidad, 1077-1078
  informalidad, 1079-1081
aridad, **278**, 321
arista (en un mapa), **908**
Aristóteles, 4, 7, 8, 12, 63, 263, 303, 353, 413, 416, 1015, 1096
Arkin, R., 1068, 1122
Arlazarov, V. L., 1121
Armstrong, D. M., 1122
Arnauld, A., 8, 664, 690, 1122
Arora, S., 99, 1122
ARPAbet, 646
arquitectura, **51**
  agente, 32
  basada en reglas, 326
  cognitiva, **325**
  de pizarra, **659**
  de subsumición, 521, **1058**
  híbrida, **1102**, 1057
  para reconocimiento del habla 30
  para tiempo real, 357
  paralela, 151
  pizarra, 659
  software, **1057**
  subsunción, 521
arrepentimiento, **672**
articulación prismática, **1027**
artificial
  edulcorante, 1083
  inseminación, 1083
  urea, 1083
  vuelo, 3
Asada, M., 1148
asar lagartos, 784
ascensión, 126, 153
  Reinicio aleatorio, **129**
ASCENSIÓN-COLINAS, **126**, 249
aserción (lógica), **287**
Ashby, W. R., *860, 1122*
asignación (en un CSP), 156
Asignación de Dirichlet latente, 973
Asimov, I., 1066, 1095, 1022
Asistente, 602
asociación , **930**
Astrom, K. J., 734, 1122
astrónomo, 607
Atanasoff, J., 16
Atkeson, C. G., 892, 1157
Atkin, L. R., 100, 1169
ATMS, *véase* sistema de gestión de verdades, suceso atómico basado en suposiciones, 534
átomos (en Lógica), 281
Audi, R., 1097, 1122
AURA, 352, 358,
Austin, G. A., *1127*
Austin, J. L., *937, 1122*
Australia, 156, 162, 172, 429
AUTOCLASS, 860
autómata, 1091, 1096
Auton, L. D., 265, 1131
autonomía, **43**, 428
auxiliar (en un CSP), **158**
AVAC, **673**, **691**
aversión al riesgo, **671**
avispa, 43
  cavadora , 43, 506
Axelrod, R., 735, 1122
axioma, **288**
  acción única, 378
    suposición de nombres únicos, 378, **378**, 403, 592
  cálculo de situaciones, 375
  de Kolmogorov, 539
  de la habitación China, 1089
  de la probabilidad, 537-541, 1114
  de la teoría
    de conjuntos, 291
    de números, 290
  del estado sucesor, **377**
  de utilidad, 666, 668
  descomposición, 667
  efecto, 375
  específico de dominio, 365
  estado sucesor, 265, 377, 414, 459
  exclusion de acción, 461

marco, 376, 379
en habla, **647**
en representación, 28, 416
problema, 376, 414, 431
inferencial, 377, 379-388
representacional, **377**, **377-378**
mundo de *wumpus*, 292
nombres únicos, 378
Peano, **290**, 303, 322
posibilidad, 375
precondición, 460
STRIPS, 468
utilizable (en OTTER), 348
axiomas
de Kolmogorov, 539
de Peano, **290**, 303, 322
del estado sucesor, 265, 414
axón, 13
Ayudante de Oficina, 602

b^* (factor de ramificación), 120
Babbage, C., 17, 208
Bacchus, F., 161, 176, 178, 566, 604, 691, 1122
Bachmann, **E G. H.**, 1115, 1122
backgammon, 196-199, 204, 211, 883
backtracking (propagación hacia atrás), 1018
cronológico, 167
dinámico, 177
inteligente, 167-169
Backus, J. W., 937, 1122
Bacon, F., 7
Baeza-Yates, R., 973, 1123, 1137
Baird, L. C. I., 733, 1175
Bajcsy, R., 1017, 1123
Baker, C. L., 34, 1123
Baker, J., 659, 1123
balance detallado, **589**
Balashek, S., 1132
Baldwin, J. M., 135, 1123
Ball, M., 1173
Ballard, B. W., 211, 214, 1123
Baluja, S., 149, 1123
Bancilhon, F., 356, 1123
banco de árboles (treebank), 973
bandido con *n*-brazos, **878**
Banerji, R., 1157
Bar-Hillel, Y., 938, 975, 1123
Barrett, A., 519, 1123
barrido con prioridad, **875**, 892
Barry, M., 606, 1144
Bar-Shalom, Y., 657, 1125
Bartak, R., 178, 1123
Bartlett, F., 15
Bartlett, P., 769, 862, 893, 1122, 1123
Barto, A. G., 150, 734, 892, 894, 1123, 1126, 1171
Barton, G. E., 943, 1123
Barwise, J., 267, 1123
Basado en cuerpo, 945
base de Herbrand, **342**
bases de datos deductivas, 352
Basye, K., 1133
Bateman, J. A., 939, 1123
Bates, E., 1135
Bates, M. A., 1172
batida, **118**
Batman, 522
Baum, E., 211, 861, 1123
Baum, L. E., 657, 859, 1123
Baxter, J., 893, 1123
Bayardo, R. J., 265, 1124
Bayerl, *S.,* 1152
Bayes, T., 546, 555, 1124
Boyer, R. S., 351, 358
Beal, D. F., 211, 1124
Bear, J., 1143
Beckert, B., 358, 1124
Beeri, C., 178, 1124
BÉISBOL, 938
Bell, C., 489, 518, 1124
Bell, D. A., 1129
Bell, J. L., 304, 1124
Bell, T. C., 973, 1175
Bellman, R. E., 2, 11, 99, 100, 146, 705, 733, 1124
Belongie, S., 862, 1018, 1124
Belsky, M. S., 1124
Bender, E. A., 1124
Bendix, P. B., 357, 1148
BENINQ, 418
Bennett, B., 1130
Bennett, E. H., 1150
Bennett, J., 1141
Bentham, J., 690, 1124
Berger, H., 13
Berger, J. O., 863, 1124
Berlekamp, E. R., 100, 1124
Berleur, J., 1091, 1124
Berliner, H. J., 210, 211, 1124
Bernardo, J. M., 1124
Berners-Lee, T., 413, 1124
Bernoulli, D., 670, 690, 1124
Bernoulli, J., 10, 555, 693
Bernstein, A., 209, 1124
Bernstein, P. L., 557, 738, 1124
Berrou, C., 604, 1124
Berry, C., 16
Berry, D. A., 892, 1124
Bertele, U., 603, 1124
Bertoli, P., 520, 1124, 1125
Bertsekas, D., 734, 894, 1116, 1125
Berwick, R. C., 1123
BESM, 209
Bestia de Hopkins, 1066
Bezzel, M., 99
Bibel, W., 354, 358, 1125, 1152
Bickford, M., 352, 1170
bicondicional, **230**
Biddulph, R., 1132
Bidlack, C., 1130
Bien y mal, 664
Bigelow, J., 18, 1165
Biggs, N. L., 176, 1125
Bilmes, J., 1164
Binder, J., 657, 860, 1125, 1166
Binford, T. O., 1018, 1125
Binmore, K., 735, 1125
bioinformática, 973
biométrica, **1004**
Birnbaum, L., 974, 1124
Biro, J., 1098, 1125
Birtwistle, G., 416, 1125
Bishop, C. M., 149, 603, 862, 863, 1125
Bishop, R. H., 63, 1134
Bistarelli, S., 176, 1125
bit (de información), 659
Bitman, A. R., 1124
Bitner, J. R., 176, 1125
BKG (programa de backgammon), 211
BLACKBOX, 464, 465, 468
Blake, A., 645, 658, 1145
Blei, D. M., 973, 1125
Blinder, A. S., 738, 1125
Block, N., 1096, 1098, 1125
bloqueo, **350**
Blum, A. L., 468, 1125
Blumer, A., 769, 1125
BNF (forma de Backus-Naur), 900, 1117
BO, *véase* optimalidad limitada
Bobick, A., 658, 1145
Bobrow, D. G., 23, 807, 974, 1125
Boddy, M., 520, 1103, 1133, 1140
Boden, M. A., 263, 1098, 1125
bolsa de palabras, **954**
Bonaparte, N., 208
Bonasso, R. P., 1150

Boneh, D., 1123
Bonet, B., 468, 521, 1125, 1126
Boole, G., 9, 263, 303, 1126
Booleano
  aleatoria, 533
  CSP, **157**
  función de tipo, 839
  lógico, *véase* lógica, proposicional, 263
  modelo de palabras clave, 953
Boolos, G. S., 358, 1126
boosting, 767
Booth, A. D., 975, 1153
Booth, T. L., 973, 1126
borde (en una escena), 987, 1000
Borel, E., 735, 1126
Borenstein, J., 1067, 1068, 1126
Borgida, A., 401, 417, 1126
borrar lista (en STRIPS), **430**
BORRAR-PRIMERO, 80, 93, 165
Boser, B., 1151
Boser, B. E., 769, 1126
Bottou, L., 1151
Boutilier, C., 415, 522, 734, 1126
Bowden, B. V., 1172
Bower, G. H., 892, 1141
Box, G. E. E, 149, 1126
BOXES, 887
Boyan, J. A., 148, 892, 1126
Boyen, X., 658, 1126
Boyle, J., 1176
BPL-RECURSIVO, 86
Brachman, R. J., 417, 419, 1126, 1152
Bradshaw, G. L., 1151
Bradtke, S. J., 892, 1123, 1126
Brady, J. M., 1159
Brafman, R. L, 522, 892, 1126
Brahmagupta, 156
Braitenberg, V., 1068, 1126
Brandeis, L., 1092
Bransford, J., 943, 1126
Bratko, 147, 357, 801, 1126, 1127
Bratman, M. E., 64
Braverman, E., 1121
Breese, J. S., 602, 1103, 1127, 1144, 1174
Breiman, L., 768, 1127
Brelaz, D., 176, 1127
Brent, R. P., 148, 1127
Bresnan, J., 938, 1127
Brewka, G., 418, 1127
Bridge Baron, 207
bridge (juego de cartas), 36
Bridle, J. S., 861, 1127
Briggs, R., 412, 1127
brillo, 984
Brin, S., 974, 1127
Brioschi, F., 603, 1124
Broadbent, D. E., 16, 1127
Broca, P., 12
Brooks, M. J., 1017, 1144
Brooks, R. A., 64, 263, 265, 521, 1058, 1060, 1067, 1068, 1127
Brouwer, P. S., 1123
Brown, J. S., 418, 809, 1122, 1152
Brown, M., 861, 1127
Brown, P. F., 968, 975, 1127
Brownston, L., 355, 1127
BRPM, 115-117
Brudno, A. L., 209, 1127
Brunelleschi, F., 1015
Bruner, J. S., 768, 1127
Brunnstein, K., 1091, 1124
Brunot, A., 1151
Bryant, R. E., 500, 520, 1127
Bryson, A. E., 26, 861, 1127
Buchanan, B. G., 26, 27, 64, 605, 768, 782, 1127, 1128, 1136, 1153
bucle de retroalimentación, 598
búfalo, 943
BUGS, 604, 859
BUILD, 418
Bulfin, R., 1164
Bundy, A., 358, 807, 1128, 1173
Bunt, H. C., *1128*
Buntine, W., 808, 1158
Burgard, W., 1067, 1128, 1137
Burns, C., 1158
Buro, M., 204, 1128
Burstall, R. M., 358, 414, 1128
Burstein, J., 1078, 1128
BUSCADORCAMINOS, 602
buscaminas, 268
búsqueda, 26, 58, 69, 97
  A*, 110-115
  A* con memoria acotada, **116**, 116-117, 147
  A* con profundización iterativa, 115, 147
  alfa-beta, 188-191, 207
  aprender a buscar, 118
  árbol, **78**
  ascensión, 126-129, 142
  B*, 210
  bidireccional, 89-91
  con información parcial, 94, 98
  con información, 82, 108, 108-119
  corte, 194
  coste uniforme, 84, 98
  de primero el mejor, 108, 145
  de profundidad limitada, 87, **87**
  de riesgos, **671**
  de rutas, 76
  de Viterbi, 654
  en tiempo real, 150, 191-196
  en un PSR, 159-169
  espacio continuo, 136-138
  estados repetidos, 91-93
  estrategia, **78**
  general, 98
  hacia adelante para planificación, 436-438
  hacia atrás, 86
    para planificación, 438, 439
  haz, **131**
    estocástico, **131**
    local, 131
  heurística, **82**, 146
  Internet, 393
  local, 125-136, 148, 177, 249-251, 264
    para PSRs, 169-171
  mediante backtracking, 86
  mejor-hipótesis-actual, 776
  memoria acotada,115-118, 147
  minimax, 185-187, 205, 208
  no informada, **82**, 82-84, 98, 99
  nodo, 78
  online,138-145, 150
  para traducciones, 970
  paralela, **151**
  política, **887**, 887-890
  primero en
    anchura, 82, 98
    profundidad, 85-87, 98
  profundización iterativa, 87-89, 98, 100, 194, 350
  recursiva, primero el mejor (RBFS), 101-102, 147
  tabú, **148**
  temple simulado, **130**
  voraz, **127**
    primero-el mejor, **108**
  vuelta atrás, 159, 167-171, 176, 1018
BÚSQUEDA, **88**, 121
  ALFA-BETA, **190**, 194
  -ÁRBOL, **79**, **81**, **82**, **85**, **93**, **98**, **108**, **111**, 145

-BACKTRACKING, 160, **160**, 167, 168, 234, 248
CON PROFUNDIDAD LIMITADA, 87
DE GRAFOS AND-OR **495**, 496, 498, 524
-GRAFO, 93, **93**, 98, 102, 108, 113, 115, 145, 151, 191, 524
-MEJOR-PRIMERO, 108
OR, 496, **496**, 525
RECURSIVA-PRIMERO-EL-MEJOR, 115, **116**
Y, 496
Bylander, T., 441, 466, 1128

caballo, 1084
cabeza (de una cláusula), **244**, 327
cadena
(en lógica), 388
(en un lenguaje), **899**
caja negra, **155**
Cajal, S., 12
cálculo, 136, 379, 384, 414
de predicados, *véase* lógica, indexación de predicados de primer orden, **318**
de situación, **373**, 373-380, 414
hedonista, 690, 691
Calera, J., 1128
Califf, M. E., 940, 1157
Calvanese, D., 417, 1128
Cambefort, Y., 64, 1141
cambio continuo, 383
Cambridge, 15
cámea-alfiler, **982**, 982-983
Cameron-Jones, R. M., 801, 1164
camino, 70, 98
aleatorio, **615**, 629
más corto, 103
Campbell, M. S., 203, 1128, 1144
Canny, J., 1017, 1068, 1128
Cantu-Paz, E., 1162 Capek, K., 1065
capa, **184**
deliberativa,1**059** Dellaert, E, 1137
ejecutiva, **1059**
reactiva, **1059**
capacidad
de sustitución (de loterías), **667**
generativa, 901, 918
capas, **841**
característica
(de un estado), **124**
(habla), **647**
Carbonell, J. G., 520, 541, 555, 556, 807, 1128, 1156, 1173
Carbonell, J. R., 807, 1128
Cardano, G., 10, 555
Carlin, J. B., 1139
CARMEL, 1067
Carnap, R., 8, 538, 604, 1128
Carpenter, M., 1133
Carrasco, R. C., 940, 1128
Carson, D., 1176
caso
acusativo, 914
dativo, 914
de condicionamiento, **564**
(lingüística), 914
nominativo, 914
subjetivo, 914
CASSANDRA, 521
Cassandra, A. R., 734, 1128, 1147
castigo perpetuo, 727
categoría, 366-373
causal, 568
(en representación), 15
kernel, **837**
lenguaje, 970
sensor, 625, 630, 656
teoría, 303
transición, **699**
causalidad, 232, 294, 549
C-BURIDAN, 520
CCD (dispositivo de carga acoplada), 981, 1019
cerebro, 19
daño, óptimo, 851
en una cubeta, 1085
estados, 1084
frente a computador, 14
potencia de computación, 14
produce la mente, 14, 1085
prótesis, 1086-1088, 1098
súper, 10
Ceri, S., 355, 1128
CES, **1061**
CFG, *véase* gramática, libre de contexto
CGP, 520
CHAFF, 249, 253, 265
Chakrabarti, P. P., 147, 1128, 1132
Chambers, R. A., 887, 892, 1156
Chan, W. P., 1017, 1129
Chandra, A. K., 355, 356, 1129
Chang, C.-L., 358, 1129
Chang, K. C., 603, 1138
Chapman, D., 467, 521, 1121, 1129
Charniak, E., 2, 27, 355, 939, 973, 975, 1129, 1139
CHART-PARSE, **913**
chatbot, 1077
Chateau Latour, 1083
Chatfield, C., 657, 1129
Chatila, R., 1066, 1158
Cheeseman, P., 10, 31, 177, 265, 605, 860, 1066, 1129, 1170
Chekaluk, R., 1113
Chen, R., 658, 1153
Cheng, J., 603, 859, 1129
Cheng, J.-F., 1155
Chervonenkis, A. Y., 769, 1173
CHESS 4.5, 100
CHESS 4.6, 210
Chickering, D. M., 211, 1142, 1149
Chien, S., 1138
Chierchia, G., 34, 938, 1129
CHILL, 936
Chinook, 204, 215
Chomsky, C., 1141
Chomsky, N., 16, 18, 901, 940, 1129
Choset, H., 1067, 1129
Chugin, 603
Chung, K. L., 556, 1116, 1129
Church, A., 10, 312, 354, 1129
Church, K., 938, 973, 1129
Churchland, P. M., 1097, 1129
Churchland, P. S., 1088, 1097, 1129
cibernética, 18
cierre de la resolución, 243
cifrado de clave pública, 351
CIGOL, 808
cilindro generalizado, **1018**
Cimatti, A., 469, 1124, 1125, 1130
cinemática, 1041
inversa, **1041**
circunscripción, **407**, 412, 417
priorizada, **408**
Clamp, S. E., 1132
Clark compleción, 418
Clark, K. L., 418, 1130
Clark, P., 808, 1130
Clark, S., 1134
Clarke, A. C., 602, 1091, 1130
Clarke, E., 358, 469, 1130
Clarke, M. R. B., 1130
clase
abierta, **904**
cerrada, **904**
de referencia, 538, 556
clasificación
(en lógica descriptiva), **401**
de documentos, **957**
clasificador Bayesiano, 550

CLASSIC, 401, 402
cláusula, 240
de Horn, 244, 348, 799
unitaria, 240, 346
cláusulas definidas, **244**, **318-319**
Clearwater, S. H., 736, 1130
CLINT, 808
Clinton,W. J., 1004
CLIPS, 355
Clocksin, W. F., 357, 1130
Clowes, M. B., 1001, 1017, 1130
CLP, *véase* programación lógica, restricción
CLP(R), 357
Club de ratios, 860
CMU, 32, 210, 658
CN2, 808
CNLP, 521
coarticulación, 650
Cobham, A., 10, 1130
Cobley, E, 937, 1130 Cocke, J., 1127
coches de carreras, 1106
coerción, 491, 492, 520
Coge el Dinero y Corre, 652
cognitiva
arquitectura, **325**
ciencia, **4**, 34
modelo, 3-4
psicología, **15**
Cohen, B., 1168
Cohen, C., 1130
Cohen, J., 356, 1130
Cohen, P. R., 30, 34, 522, 937, 1130
Cohen, W. W., 808, 1130
Cohn, A. G., 419, 1130
cola, 80
de prioridades, 108, 895, 847, 959, 960
FIFO, 82
LIFO, 85
colapso de espacio de versiones, 780
COLBERT, 1068
Collins, A. M., 807, 1128
Collins, G., 521, 1163
Collins, M., 861, 1130
Collins, M. J., 973, 1130
Colmerauer, A., 303, 356, 937, 1130
Colombano, S. P., 1153
color, 985-986
combinado de modelo Bayesiano, **952**
compilación, 332, 488
complejidad, 1109-1112
algorítmica, *véase* complejidad de Kolmogorov
de Kolmogorov, **768**
espacio, 81, 98
minería de datos, **30**
muestra, **762**
tiempo, 81, 98
complemento (de un verbo), **916**
completa, **535**, 554, 561, 564-568
COMPLETER, 909
completitud
de resolución, 241, 341-345
de un algoritmo de búsqueda, 81, 97
(de una base de datos), **403**
de un procedimiento de prueba, 335, 227, 262
teorema, **335**
componente (de distribución de mezcla), **826**
componentes nitroaromaticos, 805
COMPONER, 327
comportamiento emergente, 516, 1057
composición (de substituciones), 327, **327**
composicionalidad, 272, 919
compra en Internet, 391-397
comprensión del habla, 645
compresión, 882
comprobación
de modelo, **227**
de ocurrencias, 315, 330
hacia delante, 163-164
COMPROBAR-TV, 234, **234**
compromiso
epistemiológico, 276, 302
ontológico, 274, 302
¿COMPRUEBA-OC?, 316
COMPUESTO, 316
computación, 9
neuronal, *véase* red neuronal
computador, 16-18
ABC, 16
IBM, 203
vs. cerebro, 14
comunicacion, 897-937
con incertidumbre, 596-598
conciencia, 12, 1082, 1084, 1086, 1087, 1090
Conclusiones generales, 784
concurso de televisión, 669
condición de
monotonía, 146
terminación, 1049
condicional, 568
condicionamiento, 543
Condiciones eliminadas, **777**
Condon, J. H., 210, 1130
conducción sincronizada, **1028**
Conductismo, **15**, 18, 63
conectiva lógica, 19, 229, 281
conexión múltiple, **580**
conexionismo, **29**, *véase* redneuronal
conflicto, **517**, 517
Congdon, C. B., 1067, 1130
conjetura de Goldbach, 808
conjugado a priori, 822
conjunción (lógica), **229**
conjunto, 229
aleatorio, 600
de test, 752, **752**
difuso, 599, 605
(en lógica de primer orden), **290**, **291**
frontera, **779**
G, **779**, **780**
mágico, **326**
-S, **779**
soporte, **347**, 348
conjuntos
de respuestas, **406**
disjuntos, **367**
conmutatividad (en problemas de búsqueda), **160**
Connell, J., 1068, 1130
conocimiento
a priori, 741, **773**, 784, 794
básico, 340, 783
sincrónico, 293
co-NP-completo, 235, 264, 1111
consecuente, 229
consistencia, 401, 775
condición, 146
de camino, 176
de trabajo, 44
de una heurística, 113
Console, L., 1141
constante de Skolem, **311**, 354
construcción de nave espacial, 518
CONSTRUIR-SENTENCIA-DE-ACCIÓN, 219
PERCEPCIÓN, 219
CONSULTA, 52
consulta, **952**
(lógica), 287
contadores virtuales, 822
contenido estrecho, **1085**
CONTENIDOS, 903
contexto holístico, 1080
continuación, 331

continuidad (de preferencias), 667
contorno, 992, 1000, 1004
contrapositivo, 350
control
  bang-bang, **886**
  difuso, **600**
  lateral, 1014
  longitudinal, 1014
controlador, 63, 1052
  de PID, **1054**
  de referencia, **1052**
  P, **1052**
  PD, **1052**
controladores óptimos, **1052**
CONVENCER, 602
convención, 515
  educada, **1082**
conversión a forma normal, 336-337
convolución, 987, 1019
Conway, J. H., 1124
Cook, P. J., 1091, 1137
Cook, S. A., 10, 265, 1115, 1130
Cooper, G., 603, 859, 1131, 1169
cooperación, 512-513
coordinación, 513, 515-517
Copeland, J., 415, 1098, 1131
Copernicus, 1092, 1131
Cormen, T. H., 1115, 1131
Coronel West., 319
corrección de deletreo, **957**, 973
correcto, hacer lo, 2, 6, 1104
correlación, cruzada, 993
correo basura, 976
correspondencia
  deformable, **1008**
  emparejamiento ponderado, 1010
cortes de tiempo (en DBNs), **612**
Cortes, C., 1131, 1151
corteza cerebral, 12
cosificación, 388
COSTE-CAMINO, 82
coste total, 81
costera navegación, **1048**
COSTO-CAMINO, 78, 81
costo del camino, **70**, **71**, **98**, 156
Cournot, A., 735, 1131
Covington, M. A., 941, 1131
Cowan, J. D., 23, 860, 1131, 1175
Cowell, R., 1170
Cox, L, 657, 1066, 1131
Cox, R. T., 541, 555, 1131
CPCS, 571
CPT, *véase* tabla probabilidad condicional
Craig, J., 1068, 1131
Craik, K. J., 15, 1131
Crawford, J. M., 265, 1131
creatividad, 18
creencia, 387-390
  actualización, **409**
  Bayesianas, 604
  estado, 499, 532, 616, 713
  función, **598**
  grado de, 540
  red, *véase* propagación en redes
  revisión, **409**
  y deseo, 664-665
Cremers, A. B., 1128
Crick, E H. C., 135, 1174
criptoaritmética, 158
Cristianini, N., 861, 863, 1127
crítico (en aprendizaje), 60
Crocker, S. D., 1141
Crockett, L., 414, 1131
Croft, W. B., 974, 1163, 1172
Cross, S. E., 33, 1131
cruce, 132
crucigrama, 33, 48, 179
Cruse, D. A., 844, 997957, 1131
Csorba, M., 1134
CSP, *véase* problema de satisfacción de restricciones
cuantificador, **281**, 303
  ámbito, 938
  anidado 284-285
  en lógica, 281-285
  existencial, 283-284
  universal, 282-283
cuantización de vector, **648**
Cubo de Rubik, 99, 122, 993
cuello rígido, 547
cuerpo, 327
  (de una cláusula), **244**
Culberson, J., 148, 1131
Cullingford, R. E., 28, 1131
culto del computacionalismo, 1076
curiosidad, 871
Currie, K. W., 1138
curva, **752**
  árboles de decisión, 744-750
  basado en explicación, 786-791
  con s ocultas, 829
  conjunto, 756-760
  de ROC, **956**
  determinaciones, 791
  (de un espacio de estados), 113
  elemento, **60**
  gramática, 934-943
  heurística, 124
  incremental, 779, 783
  juego, 885
  listas de decisión, 763-765
  modelo de Markov oculto, 831
Cussens, J., 941, 1131
Cybenko, G., 861, 1131
CYC, 413-415
CYPRESS, 521

DAG, *véase* grafo acíclico dirigido
Daganzo, C., 603, 1131
Dagum, P., 603, 1131
Dahl, O.-J., 416, 1125, 1131
Dale, R., 941, 1132, 1164
DALTON, 809
damas, 21, 203-204, 211, 215
D'Ambrosio, B., 1168
Damerau, F., 1122
Daniels, C. J., 151, 1154
Dantzig, G. B., 149, 1132, 1175
daño cerebral óptimo, **851**
DARKTHOUGHT, 210
Darlington, J., 1128
DARPA, 33, 659
DART, 33
Darwiche, A., 603, 606, 1132
Darwin, C., 135, 1092, 1132
Dasgupta, P., 1132
Datalog, **319**, **352**, **355**
datos
  completos, **815**
  de atributos perdidos, **755**
Daun, B., *1133* Davidson, D., 415, 1132
Davies, T. R., 791, 807, 1132
da Vinci, L., 7, 1015
Davis, E., 413, 416, 419, 1132
Davis, G., 658, 1132
Davis, K. H., 1133
Davis, M., 248, 264, 341, 354, 1132
Davis, R., 808, 1132
Davis-algoritmo de Putnam, **248**
Dawid, A., 1151
Dawid, P., 1171
Dayan, P., 863, 893, 1132, 1157, 1167
DBN, *véase* red Bayesiana dinámica
DCG, *véase* gramática, Cláusula definida
DDB, 500
DDN, *véase* red dedecisión, dinámica
de Bruin, A., 1162
de consultas, 574
de demostración, 574

de Dombal, F. T., 556, 1132
de Finetti, B., 540, 555, 1132
de Freitas, J. F. G., 645, 1132
de Kleer, J., 177, 355, 418, 1132, 1137, 1174
de Marcken, C., 940, 943, 1132
De Morgan, A., 303, 1132
De Raedt, L., 808, 940, 1133, 1158
de Sarkar, S. C., 1128
de Saussure, F., 937, 1133
Deacon, T. W., 29, 1133
Deale, M., 519, 1133
Dean, M. E., 1160
Dean, T., 518, 657, 735, 1103, 1133
Dearden, R., 1133
Debevec, P., 1017, 1133
Debreu, G., 677, 1133
Dechter, R., 146, 176-178, 603, 1133
DECIR, 219-221, 305, 315, 410, 903
decisión
  minimax, **186**
  problema, 9
  racional, 528, 663, 686
  una sola vez, **665**
DECISION-LIST-LEARNING, 765
DECISION-TREE-LEARNING, 746, 749, 753-755, 765, 771, 775, 794, 797
decisor, **687**
declarativismo, **220**
decodificación de pila, 973
Decodificador A*, **654**
decodificadores, **1026**
decomposición (de planes), 523
DeCoste, D., 861, 862, 1133
Dedekind, R., 303, 1133
deducción
  natural, 353
  véase teorema deducción de inferencia lógica, **235**
Deep Blue, 32, 202, 203, 210
Deep Space One, 64, 464, 519
DeepThought, 202, 210
Deerwester, S. C., 973, 1133
defecto (en un plan), **510**
definición
  de candidato, **774**
  (lógica), 289
  recursiva, 800
Degradación aceptable, 718
DeGroot, M. H., 556, 863, 1133
DeJong, G., 806, 974, 1133, 1134
Del Favero, B. A., 1168
Della Pietra, S. A., 1127
Della Pietra, V. J., 1127
demodulación, 345, 357
demodulador, 348, 361
demostración de teoremas, 466, 521
Demostrador
  de Stanhope , 264
  de teoremas, 348-358
    como asistentes, 350
    de Boyer-Moore , 351, 358
    de Geometría, 21
Dempster, A. P., 598, 605, 657, 859, 1134
Den (denotación), 389
DENDRAL, 26, 27, 413
dendrita, 13
Denes, P., 658, 1134
Deng, X., 150, 1134
Denis, E, 940, 1134
Denker, J., 1151
Dennett, D. C., 1097, 1134
denotación (*grounding*), **228**
Deo, N., 99, 1134
dependencia
  de gran distancia, 926
  funcional, 791, 807
depuración, 297
  automatizada, 808
Derechos de perforación, 682
derivación, **225**
  inversa, **802**
desambiguación, 902, 925, 927-930, 937
desapego, 596
DeSarkar, S. C., 1132
desarrollo de Taylor, **1035**
Descartes, R., 7, 1015, 1096, 1134
descendiente (en redes Bayesianas), **569**
descomponibilidad (de loterías), **668**
descomposición
  de árbol, **173**
  en celdas, 1040, 1043
  exhaustiva. **367**
  jerárquica, 481
Descotte, Y., 518, 1134
descripción
  de longitud mínima, **768**, **815**
  de REAS **45**, **46**
descubrimiento científico, 768
deshilachado frente a ordenado, 29
DET-CONSISTENTE, 793, 794
  -MÍNIMA, 793, **793**
Detección
  automática, **501**
  de bordes, 987-990
    Canny, 857, 990, 1017
  facial, 1007
determinación, 792, 807, 808
  de valor, 895
  mínima, 793
DETERMINACIÓN-VALOR, 872
Devine, Y., 1173
DEVISER, 518
Devroye, L., 859, 860, 1134
día del verano, 1082
diacrónico, **293**
diagnosis, 528
  dental, 528
  médica, 295, 568, 597, 682, 1093
diagnóstico, 27, 32, 556, 603
diagrama
  de decisión binaria, 500
  influencia, *véase* red dedecisión
diamante de Nixon, 407, 408
diámetro *(de un* grafo), 87
diccionario, 25, **904**, 938
Dickinson, M. H.. 1129
Dickmanns, E. D., 1018, 1134
Dietterich, T., 769, 807, 894, 1121, 1134, 1168
diferenciación, 786
  simbólica, 361
DiGioia, A. M., 33, 1134
Digital Equipment Corporation (DEC), 28, 325
Dijkstra, E.W., 100, 1096, 1134
dilema del prisionero **721**
Dill, D. L., 1160
Dimopoulos, Y., 1138
Diofanto, 176
Diorio, C., 1164Diplomacia, 187
Dios, existencia de, 555
dirigido por los datos, **247**
DISAMBIGUATE, 902
discapacidades, 1098
discontinuidades, 987
discriminación net, 767
discurso, **930**
  coherente, 932-933
  comprensión, 930-934
  continuo, 651
diseño mecanismo, **729**, 729-732
disjuntos, 229
disparidad, 996
dispositivo de carga acoplada, 981, 1019

Dissanayake, G., 1134
distancia
de Hamming, **836**
de Mahalanobis, 836
en línea recta, **108**
entre bloques ciudad, 120
distopía, 1107
distribución
beta, 637, 821, 822
canónica, **570**
de celdas, 77
de cola pesada, **149**
de probabilidad conjunta, **535**
de subtareas, **485**
estacionaria, **589**, 619
gausiana, **1115**
mezcla, 826
normal, 1115
probit, **573**, 600, 603
distribuciones condicionales, 570-574
disyunción, **229**
divide-y-vence, 464, 660
división de símbolos, 462
Dix, J., 1127
Dixon, J. K., 1169
Dizdarevic, S., 1151 DLV, 417
Do, M. B., 1134
DOF, *véase* grado de libertad delfín, 12, 897
DOMAIN, 65
dominación
débil, **721**
(de heurística), **121**
fuerte, 721
por Pareto, **721**
dominancia
estocástica, **675**
estricta, **674**
dominante, estocástico, 690
Domingos, P., 556, 859, 1134
dominio
de aleatoria, **533**
de circuitos electrónicos, 297-302
de parentesco, 288-290
elemento, **277**
en lógica de primer orden, 277
en representación del conocimiento, 287
en un CSP, **156**
finito, 334
independencia, 428
Dones, *R.,* 355, 1146
Donskoy, M. V., 1121
Doran, C., 937, 1134
Doran, J., 146, 1134
Dorf, R. C., 63, 1134
Doucet, A., 658, 1134
Dow, R. J. F., 862, 1169
Dowling, W. F., 264, 1134
Dowty, D., 1134
Doyle, J., 63, 177, 417, 418, 691, 1134, 1155, 1174
DPLL, 248, 249, 258, 260, 264, 265, 268, 269
DPLL-SATISFACIBLE, 249
Drabble, B., 519, 1135
DRAGON, 659
Draper, D., 520, 1135, 1136
Drebbel, C., 17
Dreussi, J., 1169
Dreyfus, H. L., 414, 1080, 1104, 1135
Dreyfus, S. E., 99, 100, 733, 1080, 1124, 1135
Drucker, H., 1152
Druzdzel, M. J., 603, 1129
Du, D., 1135
du Pont, 28
dualismo, 7,1084, 1095
Dubois, D., 605, 1135
Dubois, O., 265, 1169
Duda, R. O., 556, 605, 859, 863, 1135
Dudek, G., 1069, 1135
Duffy, K., 1130
Dumais, S. T., 973, 1133, 1167
Dunham, B., 1137
DURATION, 522
Dürer, A., 1015
Durfee, E. H., 522
Durrant-Whyte, H., 1066, 1134, 1152
Dyer, M., 28, 938, 1135
DYNA, 892
Dzeroski, S., 803, 808, 941, 1131, 1135, 1148, 1151

E(cat; i) (suceso), 383
E_0 (fragmento inglés), 903
Earley, J., 938, 1135
Eastlake, D. E., 1141
Ebeling, C., 210, 1124, 1135
EBL, *véase aprendizaje* basado en explicaciones
Eckert, J., 16
Eco, U., 937, 1135
economía, 11-12, 63, 669
ecuación
de Viterbi, 947
deBellman, **705**
diferencial, 1051
ecuaciones Diofánticas, **176**
Edimburgo, 808, 1067
Edmonds, D., 19
Edmonds, J., 10, 1135
Edwards, D. J., 209, 1142
Edwards, P., 1135, 1096
Edwards, W., 691, 1173
EEG, 13
efecto, **430**
axiomas, **375**
externo, **482**
horizonte, 195
implícito, **378**
inconsistente, 452
interno, **484**
negativo, 470
positivo, 439
primario, 482
secundario, 482
efector, **1023**
efectos
condicionales, **494**
disjuntivos, **493**
EFFECT, 494
EFFECTS, 512
Egedi, D., 1134
egomovimiento, **994**
Ehrenfeucht, A., 769, 1125
Einstein, A., 1
Eiter, T., 417, 1135
ejecución, **69**
ejemplo, **742**
negativo, **744**
positivo, **744**
EKF, *véase* filtro de Kalman extendido eléctrico motor, **1029**
ELEGIR-
ATRIBUTO, 750-752
LITERAL, 800-801
elemento de actuación, 60
Elfes, A., 1158
Elhadad, M.,1135
Eliminación
-AND **237**
de asbestos, 673
de candidato, **778**
de Gauss-Jordan, **1113**
ELIMINAR-DEFECTO, 512
ELIMINATION-ASK, **579**
ELIZA, 1077, 1092
Elkan, C., 1135
Elliot, G. L., 177, 1142
Elman, J., 940, 1135

EM algoritmo, 617, 826, 825-834
  estructural, 834
EMNLP, 975
emparejamiento de patrones, **322**
empirismo, **7**
Empson, W., 939, 1135
EMV, *véase* valor monetario esperado
Encadenamiento
  hacia atrás, 245, 247, 326-335, 352
  hacia delante, 245, 244-246, 264, 318-326
Enderton, H. B., 304, 354, 1135
energía nuclear, 607
ENIAC, 16
enlace causal, **444**
ENLACES, 483
  ES-UN, 416
Enrutamiento de canal, 77
entorno, **37**, **44-51**
  agente único, 49
  artificial, 45
  clase, 50
  competitivo, 49
  continuo, **48**
  cooperativo, **49**
  de juegos, 214, 896
  determinista, **47**
  de trabajo, 44
  dinámico, 48
  discreto, 48
  episódico, 47
  estático, 48
  estocástico, 47
    gradiente, 844
  estratégico, **48**
  generador, 50
  historia, 699
  multiagente, 49, 387, 512
  observable, 47
  propiedades, 47
  secuencial, 48
  semidinámico, 48
  taxi, 45
ENUMERATE-
  ALL, 577
  JOINT, 544
ENUMERATION-ASK, 577
enunciado, **898**
envoltorio (de sitio Internet), **396**
EPAM (Perceptor y Memorizador Elemental), 767
Ephrati, E., 1150
epifenomenalismo, **1087**
epistemológico compromiso, 276, 302
época, 844
EQP, 351
equilibrio, 721, 873
  de Bayes-Nash, **729**
  de estrategia dominante, **721**
  maximin, **726**
  Nash, 722
equivalencia
  inferencial, 311
  (lógica), **235**
equivalente de certeza, **671**
Erdmann, M. A., 100, 1135
Erman, L. D., 1136
Ernst, G., 1159
Ernst, H. A., 1067, 1136
Ernst, M., 468, 1136
Erol, K., 1136
error, **820**
  (de una hipótesis), **761**
  función, 1115
  suma de cuadrados, **820**
escalada estocástica, 129
escáneres de rango, **1025**
  láser, 1025
Escarabajo estercolero, 43, 64
escarabajos, 275
escena, **981**
Escuela
  Estoica, 263, 414
  Gestalt, 1016
e spacio
  de configuración, **1040**
  de estados metanivel, **118**
  de versión, **778**, 780
  ocupado, **1041**
espacios de foco, **933**
Especificación universal, 310
espectro de masas, 26
e spectrofotometría, 985-986
esqueletización, 1040, **1046**
esquema
  de inducción matemática, 343
  (en un algoritmo genético), **133**
Esquimales, 275
Essig, A., 1140
estabilidad
  de un controlador, **1053**
  dinámica, **1028**
  estable, **1053**
  estática, **1028**
estabilizado (grafo de planificación), **452**
estables, 195, 208
estacionariedad
  (de preferentes), **702**
  (en aprendizaje PAC), **753**
Estado, 78, 81, 87, 93, 126
  actual, en búsqueda local, **125**
  conjunto, 498
  de creencia, 499
  de proceso, **383**
  del mundo, 68
  dinámico, **1024**, **1027**
  espacio, **70**, **98**
    metanivel, 118
  INICIAL, **70**, **81**, **86**, **93**, **98**, 116, 126, 130, 156, 183, 496
  intencional, **1085**
  interno, **55**
  más probable, **1047**
  repetido, 91
  restricción, **433**, 461
  terminal, 183
Estados Unidos, 15, 32, 161, 659, 682, 693, 855, 857, 1091, 1093
estándar de oro, **689**
estandarización independiente, 315, 359
estereograma, punto aleatorio 1016
estereoscopía binocular, 992, **996**, 996-1015
estiércol de insecto, 43
estilometría, **976**
estimación
  consistente, 583
  de postura, 1010-1012
  recursiva, 617
estímulo, 15
estrategia
  dominante, **721**
  (en un juego), **183**, 720
  mixta, **720**
  pura, **720**
estructura de frase, **899**, 937
estupidez natural, 398
estúpidos trucos de mascotas, 43
ETAPAS, 483
Etchemendy, J., 267, 1123
ética, 1090-1095
Etiqueta (en planes), **497**, 524
etiquetado de líneas, **1000**
etiquetas de línea, 1000
Etzioni, O., 64, 521, 1093, 1105, 1136, 1174
EU, *véase* utilidad, esperada
Euclides, 9, 1015
EURISKO, 808
Europa, 28

EVAL, 192, 194
EVALUACIÓN DE POLÍTICA 710
evaluación de rendimiento, **1109**
Evans, T. G., 23, 36, 1136
  cálculo, **380**, 380-381, 415, 921
  categoría, 421
  discreto, **383**
  en probabilidad, **575**
  evento, 363, 381-387
  generalizado, 381
  líquido, 383
evento generalizado, **381**
Everett, B., 1066, 1126
evolución, 36, 135, 275
  automática, 25
  máquina, 25
exacta, **1044**
excepciones, 364
exclusión mutua, **451**
EXPAND, 81, 93, 116
  -GRAPH, 455
expansión (de estados), **78**
expectación-maximización, *véase* EM
EXPECTIMINIMAX, 718
expectiminimax, 199, 211
  complejidad de, 199
explicación, **411**, 787
  más probable, 656
exploración, 43, 138-146, 892
explorador planetario, **901**
explosión combinatoria, 25
EXTENDER, 234, 250
extensión
  (de la teoría por defecto), 408
  (de un concepto), 774
  (de un enlace causal), 509
  excepcional, **195**
  iterativa, **147**
EXTENSOR, 910, 909, 912, 913
externalidades, **730**
EXTRAER-SOLUCIÓN, 454, 455, 459

factor
  de certeza, *27, 598*
  de cuantización, **647**
  de descuento, 702, 733
  de ramificación, 81, 790
    efectivo,120, 146, 190
  (en eliminación de), **577**
factorización, 240, 338
Fagin, R., 177, 416, 1124, 1136
Fahlman, S. E., 417, 1136
fallo transitorio, **637**
falso positivo, **775**
Farrell, R., 355, 1127
fase redundante, 509
FASTFORWARD, 468
FASTUS, 962, 974
Faugeras, O., 1017, 1136
Fearing, R. S., 1068, 1136
Featherstone, R., 1068, 1136
Feigenbaum, E. A., 26, 27, 34, 413, 768, 1128, 1136, 1153
Feinstein, M. H., 1171
Feldman, J., 34, 692, 1136
Feldman, R., 807, 1171
Fellbaum, C., 938, 1136
Feller, W., 1116, 1136
Felner, A., 148, 1150
Feng, C., 808, 1158
Feng, L., 1067, 1126
fenomenología, 1082
Fermat, P., 10, 555
Ferraris, P., 520, 1136
fertilidad, 969
FETCH, 316, 359
FF (planificador delantero rápido), 468
Figura de discurso, 928
fijación, **997**
Fikes, R. E., 64, 303, 357, 417, 466, 519, 521, 807, 1066, 1126, 1136, 1146
filo de Megara, 263, 414
filosofía, 6, 9, 63, 1075-1098
FILTER, 349
filtrado, 409, 616, 617-619, 656, 713
  de partículas, 643, **643**, 656
filtro
  de Kalman extendido (EKF). 633, 1036
  Gausiano, **987**
FIN, 910
Findlay, J. N., 414, 1136
FIND-
  PURE-SYMBOL,250
  TRANSFORM, 1011, 1012, 1020
  UNIT-CLAUSE, 250
Finney, D. J., 603, 1137
Firby, J., 1060, 1137
Firby, R. J., 519, 522, 1133, 1137
FIRST, 80, 316, 327
Fischer, M. J., 414, 1137
Fisher, R. A., 555, 1137
física cualitativa, 371, **418**
Fix, E., 860, 1137
FIXED-LAG-SMOOTHING, **627**
Flannery, B. P., 1163
flujo, **373**, **386-387**, **926**
  óptico, **993**, 1014, 1016
FMP, *véase* planificación, movimiento fino
fMRI, 13
foco, 808, 983
  de expansión, **994**
Fogel, D. B., 150, 1137
Fogel, L. J., 150, 1137
FOIL, 798, 800, 808, 809, 810
FOL, *véase* lógica, primer order
FOL-BC-ASK, 326, 327, 330
FOL-FC-ASK, 320, 321, 326
fonema, **646**
fonología, **646**
FOPC, *véase* cálculo de predicados
Forbes, J., 892, 1137
FORBIN, 518
Forbus, K. D., 355, 418, 1137
Ford, K. M., 1096, 1137
Forestier, J.-P., 894, 1137
Forgy, C., 355, 1137
forma, **992**
  contexto, **1009**
  cuasi-lógica **923**, 938
  de Backus-Naur (BNF), 900, 1117
  de sombreado, 1016
  de textura, 1016
  intermedia, **922**
  Kowalski, 336
  normal
    conjuntiva, 241, 336-338
    de Chomsky, **951**
    de Clark, 403
formación
  con conjuntos de test, 752
  conjunto, **747**, 752, 766
  curva de, **850**
  ponderada, **758**
  replicada, **758**
formalismo gramatical, 901
formular, buscar, ejecutar, 68
FORMULAR-
  META, 69
  PROBLEMA, 69
Forrest, S., 150, 1147
Forsyth, D., 1017, 1137
Fortescue, M. D., 275, 1137
Fortmann, T. E., 657, 1123
FORWARD, 629
  -BACKWARD, **621**
Foster, D. W., 976, 1137
fotogrametría, 1016
fotometría, **983**, 983-985

Fourier, J., 176, 1137
Fox, D., 1067, 1128, 1137
Fox, M. S., 468, 518, 1137
Frakes, W., 973, 1137
Francis, S., 973, 1137
Franco, J., 265, 1137
Frank, M., 1140
Frank, R. H., 1091, 1137
Frankenstein, 1093
frase, 900
  nominal, **900**
  verbal, 900
frecuentismo, 538
FREDDY, 77, 520, 1067
Fredkin Prize, 210
Freeman, W., 1176
Frege, G., 9, 263, 303, 353, 1137
fresas, disfrute, 1077
Freuder, E. C., 177, 178, 1137, 1138
Freund, Y., 768, 1138
Friedberg, R. M., 25, 150, 1138
Friedman, G. J., 149, 1138
Friedman, J., 768, 863, 1166
Friedman, N., 658, 859, 1126
Fristedt, B., 892, 1124
FRITZ, 203
frontera, **79**
Frost, D., 178, 1133
FRUMP, 974
Fry, D. B., 658, 1138
Fuchs, J. J., 419, 1138
Fudenberg, D., 735, 1138
FUF (programa de generación), 939
Fukunaga, A. S., 419, 1138
función, **108**, 119-124, 273, 428
  acción-valor, 868
  aproximación, 882
  aproximadora, 882
  búsqueda, *véase* búsqueda,
  de activación, 839
  de evaluación, **108**, **182**, **192-194**
    lineal, 124
    precisión, 192
  de exploración, 879, 881
  de navegación, **1048**
  densidad probabilidad acumulativa, 1115
  de paridad, **746**
  de rentabilidad, 183
  de Skolem, **337**, 354
  de unidades, **369**
  de utilidad, 59
  heurística de la línea recta, 108
  lineal ponderada, **193**
  logística, **839**
  Manhattan, **120**
  mayoría, **746**, 842
  mínimos conflictos, **169**
  objetivo, **126**, **156**
  sigmoide, **573**, **839**
  símbolo, **278**, **281**, **302**
  softmax, **888**
  sucesor, **70**, 156, 183
  total, 278
  valor, **673**
    aditivo, **678**
    menos restringido, **163**
  valores restantes mínimos, 162, 176
funcionalismo, 63, **1083**, 1088, 1096
fundamental, 400
Fung, R., 1138
Furey, T., 1127
Furnas, G. W., 1133
Furst, M., 468, 1125
FV (frase verbal), 900

Gabor, Z. Z., 693
Gaifman, H., 604, 1138
galardón Turing, 1115
Gale, W. A., 973, 1129
Galileo, G., 1, 61, 804
Gallaire, H., 355, 1138
Gallier, J. H., 264, 304, 354, 1134, 1138
Gallo, G., 99, 1138
Gamba, A., 861, 1138
Gamberini, L., 1138
García, P., 940, 1160
Garding, J., 1017, 1138
Gardner, M., 264, 1138
Garey, M. R., 1115, 1138
Garfield, J. L., 34, 1171
GARI, 518
Garrett, C., 1123
Gaschnig, J., 146, 152, 177, 1135, 1138
Gasquet, A., 1138
Gasser, R., 204, 1138
Gat, E., 1068, 1138
Gauss, C. F., 99
Gauss, K. F., 176, 656, 1138
Gaussiana condicionada, **573**
Gawande, A., 1093, 1139
Gawron, J. M., 659, 1147
Ge, N., 939, 1139
Gee, A. H., 1132
Geffner, H., 468, 519, 520, 1126, 1142
Geiger, D., 602, 1139, 1142
Gelatt, C. D., 1148
Gelb, A., 657, 1139
Gelernter, H., 21, 357, 1139
Gelfond, M., 417, 1139
Gelman, A., 863, 1139
Geman, D., 603, 1139
Geman, S., 603, 1139
generación (de estados), **78**
Generador de Consejos, 22, 27
generalización, **776**, 777
generalizada, 789
Generalizado, **313**,313
GENERAR-DESCRIPCIÓN, 603
género natural, **370**
Genesereth, M. R., 63, 100, 221, 303, 304, 335, 341, 358, 1072, 1139, 1170
Gentner, D., 807, 1139
Gentzen, G., 353, 1139
Georgeff, M. P., 522, 1139
Gerbault, F., 1140
Gerevini, A., 468, 1139
Germann, U., 1139
Gershwin, G., 1139
Ghahramani, Z., 658, 1139, 1146, 1165
Ghallab, M., 466, 468, 519, 1139
Ghose, S., 1128
Giacomo, G. D., 415, 1139
GIB, 205
Gibson, J. J., 1016, 1139
Gilks, W. R., 604, 859, 1140
Gilmore, P. C., 354, 1139
Gini, M., 520,1160
Ginsberg, M. L., 177, 205, 522, 606, 1132, 1140, 1170
Gittins, J. C., 878, 892, 1140
Giunchiglia, E., 520, 1136
Givan, R., 938, 1155
Glanc, A., 1065, 1140
GLAUBER, 809
Glavieux, A., 1124
Glover, F., 148, 1140
Glymour, C., 1169
Go (juego), 204
Go4++, 205
Gödel, K., 9, 264, 335, 354, 1140
Goebel, J., 860, 1140
Goebel, R., 63, 1163
Goemate, 205
GOFAI, *véase good old-fashioned* AI
Gold, B., 1140
Gold, E. M., 768, 940, 1140

Goldberg, D. E., 1162
Golden, K., 521, 1140
Goldman, N., 939, 1140
Goldman, R., 939, 1129, 1140, 1174
Goldszmidt, M., 859, 1126, 1138
GOLEM, 808
Golgi, C., 12
GOLOG, 415, 1060, 1061
Gomard, C. K., 1146
Gomes, C., 149, 1140
Go-Moku, 204
Good, 1. J., 209, 602, 1094, 1140
Gooday, J. M., 1130
Goodman, D., 32, 1140
Goodman, N., 415, 806, 1140, 1152
Goodnow, J. J., 1127
Gordon, M. J., 303, 1140
Gordon, N. J., 658, 1140
Gorry, G. A., 556, 1140
Gottlob, G., 178, 356, 1128, 1140
Gotts, N., 1130
GPS, *see* Sistema Global de Posicionamiento
gradiente
  de textura, **997**, 1016
  descendente, 130
grado de
  creencia, **529**
    valor de intervalo, 596
  verdad, 274
graduación recíproca (en IR), 956
gráficas felices, 752
gráficos por computador, **980**
grafo, **908**
  acíclico dirigido (DAG), 562, 602
  de visibilidad, **1067**
  de Voronoi, 1047
  existencial, **397**
  Y-O, **245**
GRAFO-PLANIFICACIÓN-INICIAL, 454
Graham, S. L., 938, 1141
gramática, **899**, 1117
  adjunta a árboles (TAG), 938
  atributo, **937**
  aumentada, 914-919
  categorial, 938
  cláusula definida (DCG), **915**, 936
  dependencia, 938
  dependiente del contexto, 901
  estructuras de frase, 936
  formal, 903
  independiente de contexto, **901**, 936
    probabilista (PCFG), **949**, 949-951, 973
  inducción, **934**, 940
    gramatical, 934-943
  inglesa, 903-905
  léxico-funcional (LFG), 938
  probabilista, 945-952
  recursivamente enumerable, 901
  regular, 901
  sistémica, 927
  universal **940**
Gran Premio, 202
GRAPHPLAN, 450, 454- **454**, 457, 458, 463-469, 472, 520
Grassmann, H., 303, 1141
Grayson, C. J., 590, 1141
Green, B., 938, 1141
Green, C., 22, 303, 352, 354, 358, 414, 466, 1141
Greenbaum, S., 1164
Greenblatt, R. D., 209, 1141, 1158
Greenstreet, M. R., 358, 1148
Greiner, R., 807, 1129, 1141
Grice, H. P., 937, 1141
Griegos, 263, 353, 412, 415
GRL, 1068
Grosof, B., 807, 1166
Grosz, B. J., 730, 736, 933, 940, 1141, 1145
Grove, A., 556, 691, 1122
Grove, W., 1077,1141
Grumberg, O., 469, 1130
Grundy, W., 1127 GSAT, 265
Gu, J., 177, 264, 1135, 1141, 1170
Guard, J., 351, 357, 1141
Guha, R. V., 413, 1152
Guibas, L. J., 1141
Gus, 974
Gutfreund, H., 1121
Guy, R. K., 100, 1124
Guyon, I. M., 1126
Guyon, I., 1152
Gyorfi, L., 1134

Haas, A., 416, 1141
habitación china, 1088-1090, 1098
HACER
  COLA, 80
  -NODO, 81, 86, 93, 116, 126, 130
HACKER, 467
Hacking, 1., 557, 1141
Hähnel, D., 1128
Haken, W., 176, 1132
HAL 9000 computador, 529
Hald, A., 557, 1141
Hale, J., 1139
Halpern, J.Y., 556, 604, 1122, 1136, 1141
Halpin, M. P., 1140
Hamming, R. W., 557, 1141
Hammond, K., 520, 1141
Hamscher, W., 64, 1141
Handschin, J. E., 658, 1141
Hanks, S., 1135, 1136
Hanna, F. K., 809, 1165
Hansard, **967**, 977
Hansen, E, 264, 1141
Hansen, E., 148, 520, 734, 1141, 1177
Hanski, L, 64, 1141
Hansson, O., 148, 152, 1141
Harada, D., 1159
Haralick, R. M., 176, 1142
Hardin, G., 736, 1142
Harel, D., 355, 414, 1129, 1142
Harman, G. H., 1097, 1142
HARPY, 148, 658
Harrison, M. A., 938, 1141
Harsanyi, J., 735, 1142
Harshman, R. A., 973, 1133
Hart, P. E., 146, 556, 859, 863, 1135, 1136, 1142
Hart, T. P., 209, 1142
Hartley, R., 1017, 1142
Harvard, 672
Haslum, P., 519, 1142
Hastie, T., 862, 863, 1138, 1142
Haugeland, J., 2, 34, 1080, 1098, 1142
Haussler, D., 769, 808, 861, 862, 1123, 1125, 1127, 1142
Havelund, K., 352, 1142
Hayes, P. J., 34, 414, 415, 417, 418, 1096, 1137, 1142 1155
Hayes-Roth, E, 1136
HEARSAY-II, 659
Hearst, M. A., 975, 1160, 1167
Heath Robinson, 16
Heawood, P., 1079
Hebb, D. O., 19, 23, 891, 1142
hecho, 244
Heckerman, D., 31, 32, 598, 602, 605, 692, 859, 1129, 1142, 1144, 1167, 1170
Heim, 1., 938, 1142
Heinz, E. A., 210, 1142
helado, 531
Held, M., 148, 1142

Helmert, M., 469, 1143
Helmholtz, H., 14, 1016
Hempel, C., 8
Henderson, T. C., 165, 177, 1157
Hendler, J., 519, 1121, 1124, 1136
Hendrix, G. G., 400, 1143
Henrion, M., 64, 603, 692, 1143, 1144, 1163
Henry, O., 487
Henzinger, M., 1169
Henzinger, T. A., 63, 1143
Hephaistos, 1066
Herbrand, J., 264, 312, 342, 354, 1143
herencia, 366, **366**, 398, 424
Herskovits, E., 859, 1131
heurística, 145
  admisible, **111**
  compuesta, 122
  del valor menos restrictivo, 163
  grado, 162, 176
  independiente del dominio, 429
  para planificación, 441-454
  suprimir listas vacías, **441**
Hewitt, C., 355, 356, 1143
Hex, 212
hexápodo, **1056**
Hierholzer, C., 150, 1143
Hilbert, D., 9
Hilgard, E. R., 892, 1143
Hingorani, S. L., 1131
Hintikka, J., 416, 1143
Hinton, G. E., 29, 149, 861, 863, 1143, 1165
hiperparámetro, 821
HipNav, 33
hipótesis, **651**
  aproximadamente correcto, 761
  de mundo abierto, 459, 500
  de Sapir-Whorf, 275
  extensión, 774
  nula, **754**
HIPÓTESIS-RED-NEURONAS, 845, 849
Hirsh, H., 806, 1143
Hirst, G., 939, 1143
Hitachi, 489
HITECH, 210
hMAP (MAP hipótesis), 814
hML (ML hipótesis), 815
Ho, Y.-C., 26, 861, 1127
Hoane, A. J., 202, 1128
Hobbes, T., 7
Hobbs, J. R., 932, 933, 939, 974, 1143
Hockey, B. A., 937, 1134
Hodges, J. L., 860, 1137
Hoff, M. E., 23, 870, 891, 1175
Hoffman, J., 468, 1149
Hoffmann, J., 468, 1143
Hogan, N., 1068, 1143
Holland, J. H., 149, 1143, 1157
Holldobler, S., 414, 1143
Hollerbach, J. M., 1068, 1136
Holloway, J., 1158
Holzmann, G. J., 352, 1143
hombre primitivo, 784
homeostático, 860
Homo sapiens, 1
homófono, **645**
Hon, H., 1144
Honavar, V., 940, 1161
Hong, J., 1156
Hood, A., 12, 1143
Hopcroft, J., 1115, 1121, 1167
  red Hopfield, **862**
Hopfield, J. J., 29, 862, 1143
horizonte, 982
  finito, **701**
  infinito, **701**
Horn, A., 244, 262, 263, 1144
Horn, B. K. E, 1017, 1018, 1144
Horning, J. J., 1144
Horowitz, E., 99, 1144
Horowitz, M., 1160
Horrocks, J. C., 1132
Horswill, L, 1060, 1068, 1144
Horvitz, E. J., 31, 64, 602, 606, 692, 1103, 1144, 1167
Hovel, D., 602, 606, 1144
Hovy, E., 939, 1144
Howard, R. A., 679, 691, 692, 733, 1144, 1156
Howe, A., 1139
hoyos sin fondo, 221
HSCP, 520
Hsu, F.-H., 202, 210, 1128, 1144
HTML, 963
HTN, *véase* red de tareas jerárquica
Huang, T., 658, 1144
Huang, X. D., 659, 1144
Hubel, D. H., 1144
Huber, M., 1066, 1130
Huddleston, R. D., 937, 1144
Huffman, D. A., 23, 1001, 1017, 1145
Huffman, S., 1130
Hughes, B. D., 142, 1145
HUGIN, 602, 658
Huhns, M. N., 64, 1145
Hume, D., 7, 1145
Hunsberger, L., 730, 1145
Hunt, E. B., 768, 1145
Hunter, L., 860, 1145
Hurwicz, L., 736, 1145
Hutchins, W. J., 975, 1145
Huttenlocher, D. P., 1017, 1145
Huygens, C., 555, 735, 1145
Hwa, R., 973, 1145
Hwang, C. H., 937, 1145
Hyun, S., 1133

I.A. fuerte AI, **1075**, 1081-1090
i.i.d. (independientemente e idénticamente distribuido), **817**
IBM, 20, 21, 32, 202, 659, 969
ID3, 808
IDA*, *véase* búsqueda, profundización iterativa A*
identificación del hablante, 647
  especialización, **776**
identificar en el límite, **765**
IEEE, 413
ignorancia, 596, 598
  práctica, **529**
  teórica, **529**
igualdad (en lógica), 286
IJCAI (International Joint Conference on AI), 35
  formación, 981-986, 1014
ILP, *véase* programación lógica,
  inductiva imagen, **983**
  procesamiento, 1015
  recuperación, **1004**
  segmentación, 990-991, 1005
¿IMPLICACIÓN-EHD-LP?, 245, **246**
¿IMPLICACIÓN-EN-TV?, 234, 248, 250, 254, 267
impulso diferencial, 1028
incertidumbre, 27, 364, **527**, 527-560, 598, 1081
  de identidad, **593**
  enfoque basado en reglas, 596
  razonamiento con, *véase* razonamiento, relacional, **593**
  resumen 529
  y tiempo 612-616
incertidumbre de control, 1049
inclinación, **992**
incorporación, **902**
independencia **546**, 544-546, 549, 554
  absoluta 546
  condicional, **549**, **552**, **553**, **558**, **568-574**, **601**, **620**

de parámetros, 822
de preferencias, **677**, 694
de sub-objetos, **440**
de utilidad mutua (IMU), **679**
preferencial mutua **(IMP)**, **677**
indeterminación, **490**
ilimitada, **490**
indexación semántica latente, 973
indexar, **316**, 316-317
India, 18, 176, 412
indicadora, **846**
indiferencia, principio de, 538, 555
indirecto, **898**
individualización, **372**
inducción, **7**, 742
constructiva, **798**
matemática, 9
inductivo, 742-744, 766
basado en
conocimiento, **786**, 795, 806
instancia, 835, **835**
Bayes simple, 818
conocimiento en, 782-786
de arriba a abajo, 798-801
diferencia temporal, 872-875
estadístico, 811-815
funciones
de utilidad, 868
separables linealmente, 843-846
lógico, 773-782
MAP, 814-815
máxima verosimilitud, 815-820
no paramétrico, 835
no supervisado, **650**, 826-829
nuevos predicados, 804
PAC, 761, 769, 791
para buscar, 118-119
parámetro, **815**, 821-823, 834
pasivo, **869**
problema del restaurante, 745
Q, 880, 887
red neuronal, 748
refuerzo, 867-896
ruido, 753-755
supervisado, 1005
Indyk, P., 860, 1145
inferencia, **219**, 309
inductiva pura, **742**
probabilista, **541**, 541-544, 561
procedimiento, 297
regla, 237, 262
temporal, 616-624
INFIERE, 349, **349**
influencia causal, 566
información
extracción, 961-964, 972
ganancia, **751**, 754
recogida, **43**, 1048
recuperación, 393, 952-961, 973
teoría, **750**, 750-752, 767
valor, **682**, 692
INFORMATION-GATHERING AGENTE, **686**
Informe
Alvey, 28
Lighthill, 25, 28
ingeniería ontológica, 364, 363-366
Ingerman, P. Z., 937, 1145
inglés, 25, 36
fragmento, 903
Inglés Caterpillar, 964
INICIAR, 911
Iniciativa de Mente Abierta, 413
INIT, 907
Inoue, K., 1145
INSERT, 80, 81, 93
-ALL, 80, 81, 93
inspector de higiene de restaurantes, 202
instancia
intencional, 1097
típica, 370
Instanciación Existencial, 310
integrabilidad, **999**
integración simbólica, 782
intelectual humana, 1
inteligencia, 1, 37
artificial, 1, 1-1107
aplicaciones, 32-33
asociaciones, 35
bases (fundamentos), 6-19, 881
como campo universal, 1
como diseño de agente racional, 5
conferencias, 35
definición, 2
en tiempo real, **1103**
filosofía de la, 1075-1098
futuro de, 1106-1107
historia, 19-32
lenguaje de programación, 21
objetivos de, 1104-1106
posibilidades, 1075-1081
potente, **1106**
revistas, 35
simple, **1105**
subcampos, 1
intención
conjunta, **517**
(discurso), **900**
intencionalidad, 1082, 1096, 1097
intercalar (búsqueda y acción), **97**
interlingua, 965
Internet, 30
INTERPLAN, 467
interpretación, **279**
pragmática, **900**, 925-926
semántica, **900**, 919-925, 938, 941
INTERPRETAR-ENTRADA, 54
interreflexiones, 1000
intervalo, 380, 384-385
de protección, *véase* enlace causal
temporal, 415
Intille, S., 1145
intratabilidad, **10**, **25**, **35**
Introducción Existencial. **359**
introspección, 4, 15
investigación operativa, 11, 63, 100, 147, 519
Invierno de la IA, 29
Inza, L, 1151
IPEM, 521
IPL, 20
IPP, 468
irracionalidad, 2, 666, 691
Isard, M., 645, 658, 1145
ISIS, 518
Israel, D., 1143
Italia, 555
Italiano, 965
ITEP programa de ajedrez, 209
ITERACIÓN
DE POLÍTICA, **710**
de valores, **704**, 704-09, 733
-VALOR, 706
ITERATIVE-DEEPENING BÚSQUEDA, **88**, 121
ITOU, 808
IXTET, 468

Jaakkola, T., 893, 1145, 1146, 1167
Jackel, L., 1151
Jacquard Ioom, 17
Jacquard, J., 17
Jaffar, J., 357, 1145
Jaguar, 518
Jahr, M., 1139
James, H., 15
James, W., 15
Japón, 28
jaque mate, 91
Jáskowski, S., 354, 1145
Jaumard, B., 264, 1141

Jeavons, P., 178, 1161
Jefferson, G., 1082, 145
Jeffrey, R. C., 358, 555, 691, 1126, 1145
Jeffreys, H., 973, 1145
Jelinek, F., 659, 973, 1127, 1145, 1146
Jenkin, M., 1069, 1135
Jennings, H. S., 15, 1146
Jensen, E, 602, 1122
Jensen, F. V., 602, 606, 1122, 1146
jerarquía
  de generalización, **782**
  taxonómica, 28, 367
Jespersen, O., 937, 1146
Jevons, W. S., 807, 1146
Jiménez, P., 520, 1146
Joachims, T., 861, 974, 1146
Johnson, C. R., 64, 1128
Johnson, D. S., 1115, 1138
Johnson, M., 939, 943, 1126, 1135, 1151
Johnson, W. W., 99, 1146
Johnson-Laird, P. N., 34, 1146
Johnston, M. D., 519, 1146, 1157
Jones, M., 1018, 1173
Jones, N. D., 807, 1146
Jonson, A., 32, 64, 519, 1146
Jordan, M. 1., 604-606, 658, 861, 889, 893, 1125, 1139, 1145, 1146,1158, 1159, 1167, 1170
Joshi, A. K., 938, 940, 1141 1146, 1167
Joskowicz, L., 418, 1166
Joslin, D., 468, 1146
Jouannaud, J.-P., 357, 1146
Joule, J., 804
JTMS, *véase* sistema de mantenimiento de verdad, basado en justificación,
Juang, B.-H., 659, 1164
Judd, J. S., 862, 1146
juego, 11, **181**
  bridge, 200
  carta, 200
  constante-suma, **723**
  de apuesta, 11, 540, 668
  de coordinación, 723
  de suma cero, **723**
  del Almanaque, 692
  dados, 200
  de posibilidad, 196-202
  dragaminas, 264
  frente a naturaleza, **494**
  información parcial, **728**
  inspección, **719**
  jugar, 181-182, 207
  multijugador, 186-187
  Otelo, 182, 204
  póker, 557
  programas, 202-205
  reiterativo, **726**
Juels, A., 149, 1146
Juicio humano, 605, 672
Julesz, B., 1016, 1146
Jurafsky, D., 34, 659, 940, 975, 1146
justificación, **406**, 597

Kadane, J. B., 735, 1147
Kaelbling, L. P., 265, 644, 735, 894, 1068, 1128, 1133, 1147
Kager, R., 939, 1147
Kahneman, D., 2, 568, 672, 1147, 1173
Kaindl, H., 147, 1147
Kalah, 209
Kalman, filtro, 611, **627**, 627-635, 656, 657
  conmutación, **634**
Kalman, R., 627, 657, 1147
Kambhampati, S., 468, 519, 520, 1134, 1147, 1159
Kameya, Y., 605, 1167
Kameyama, M., 1143
Kan, A., 99, 1151
Kanade, T., 33, 995, 1016, 1134, 1172
Kanal, L. N., 147, 606, 1147, 1150, 1159
Kanazawa, K., 579, 735, 860, 1125, 1133, 1147, 1166
Kanefsky, B., 10, 1129
Kanoui, H., 356, 1130
Kant, E., 355, 1127
Kaplan, D., 416, 1147
Kaplan, R., 913, 938, 1125, 1155
Karmarkar, N., 149, 1147
Karmiloff-Smith, A., 1135
Karp, R. M., 10, 99, 148, 1115, 1142, 1147
Karypis, G., 1170
Kasami, T., 938, 1147
Kasparov, G., 32, 202, 203, 1147
Kasper, R. T., 939, 1123, 1147
Kassirer, J. P., 1140
Kaufmann, M., 358, 1147
Kautz, D., 1133
Kautz, H., 468, 1140, 1147, 1168
Kavraki, L., 1068, 1147
Kay, A. R., 1160
Kay, M., 659, 965, 975, 1125, 1147
Kaye, R., 264, 1147
KB, *véase* base de conocimiento
KBIL, *véase* aprendizaje inductivo, basado en conocimiento
k-DL (lista de decisión), 764
k-DT (árbol de decisión), 764
Keane, M. A., 1150
Kearns, M., 734, 769, 1147
Kedar-Cabelli, S., 807, 1157
Keene, R., 32, 1140
Keeney, R. L., 672, 678, 691, 1148
Kehler, A., 939, 1148
Keil, F. C., 4, 34, 1097, 1175
Keim, G. A., 33, 1153
Keller, R., 806, 1157
Kelly, J., 860, 1129
Kemp, M., 1015, 1148
Kempe, A. B., 1079
Kenley, C. R., 603, 1168
Kepler, J., 1015
Kern, C., 358, 1148
Kernel, función, 837, 854
  polinomial, 854
Kernighan, B. W., 99, 1153
Keynes, J. M., 555, 1148
Khatib, O., 1068, 1148
Khorsand, A., 147, 1147
KIDS, 358
Kietz, J.-U., 808, 1148
Kilimanjaro, 897
Kim, J. H., 602, 1148
King, R. D., 805, 1148, 1170
Kirby, M., 1018, 1169
Kirchner, C., 357, 1146
Kirkpatrick, S., 149, 157, 265, 1148
Kirman, J., 1133
Kirousis, L. M., 1017, 1148
Kitano, H., 1067, 1148
Kjaerulff, U., 658, 1148
Kleer, J. D., 63, 1141
Klein, D., 973, 1148
Kleinberg, J. M., 973, 1148
Klempner, G., 606, 1158
KL-ONE, 417
Knight, K., 2, 974-977, 1139, 1148, 1164, 1176
Knoblock, C. A., 99, 466, 1128, 1139, 1148
Knuth, D. E., 209, 357, 937,1141, 1148
Koditschek, D., 1068, 1148

Koehler, J., 468, 1149
Koenderink, J. J., 1016, 1017, 1149
Koenig, S., 100, 150, 734, 1067, 1149, 1169
Kohn, W., 357, 1149
Koller, D., 556, 605, 659, 735, 860, 974, 1122, 1125, 1126, 1144, 1147, 1149, 1157, 1166
Kolmogorov, A. N., 555, 657, 768, 1149
Kolodner, J., 28, 807, 938, 1149
Kondrad, G., 177, 178, 1149
Konolige, K., 416, 417, 522, 1068, 1137, 1149
Koopmans, T. C., 734, 1149
Koren, Y., 1068, 1126
Korf, R. E., 100, 148, 150, 211, 467, 1149, 1150, 1162
Kortenkamp, D., 1069, 1130, 1150
Koss, E, 1066, 1130
Kotok, A., 209, 1150
Koutsoupias, E., 148, 265, 1150
Kowalski, R., 303, 328, 336, 356, 357, 414, 415, 418, 1150, 1166, 1173
Koza, J. R., 150, 1150
KPML (programa de generación), 939
Kramnik, V., 203
Kratzer, A., 938, 1142
Kraus, S., 522, 1150
Kraus, W. F., 1153
Krauss, P., 604, 1168
Kripke, S. A., 416, 1150
Krishnan, T., 859, 1156
Krovetz, R., 974, 1150
Kruppa, E., 1016, 1150
KRYPTON, 417
Ktesibios, 17
Kucera, H., 973, 1137
Kuehner, D., 356, 1150
Kuhn, H. W., 736, 1150
Kuhns, J.-L., 973, 1154
Kuijpers, C., 1151
Kuipers, B. J., 1067, 1150
Kukich, K., 974, 1150
Kulikowski, C. A., 769, 1174
Kumar, P. R., 63, 1150
Kumar, V., 148, 178, 1147, 1150, 1159, 1170
Kuniyoshi, Y., 1148
Kuper, G. M., 356, 1150
Kurzweil, R., 2, 1094, 1150
Kyburg, H. E., 556, 1151
Ladkin, P., 415, 1151
Ladner, R. E., 414, 1137
Lafferty, J., 974, 1151
Laguna, M., 148, 1140
Laird, J. E., 31, 325, 355, 519, 708, 971, 1146, 1151
Laird, N., 657, 860, 1134
Laird, P., 1157
Lakemeyer, G., 1128
Lakoff, G., 413, 939, 1151
Lamarck, J. B., 135, 1151
Lambertian, superficie, 985, 999
La Mettrie, J. O., 1096, 1151
La Mura, P., 691, 1151 LCF, 303
Landauer, T. K., 1133
Langley, P., 809, 1151
Langlotz, C. P., 1144
Langton, C., 149, 1151
Lansky, A. L., 522, 1139
Lanzamiento de moneda, 597, 598, 694, 750
Laplace, P., 10, 538, 556, 595, 1151
Lappin, S., 1151
Lari, K., 1151
Larkey, P. D., 735, 1147
Larrañaga, P., 149, 153, 1151
Larsen, B., 602, 1151
Larson, G., 783
Laruelle, H., 468, 519, 1139
Lassez, J.-L., 357, 1145
Lassila, O., 1124
latente, **825**
Latombe, J.-C., 518, 1068, 1134, 1147, 1151, 1177
Laugherty, K., 938, 1141
Lauritzen, S., 603, 606, 692, 860, 1151, 1160, 1170
Lavrac, N., 803, 808
Lavraé, N., 1151, 1156
LAWALY, 519
Lawler, E. L., 99, 147, 480, 1151
Lazanas, A., 1068, 1151
Lazo abierto, **70**
Le Cun, Y., 862, 1151
Leacock, C., 1128
LEANTAP, 358
Leaper, D. J., 556, 1132
Leass, H. J., 939, 1151
LeCun, Y., 862, 1152
Lederberg, J., 27, 1136, 1153
Lee, K.-F., 659, 1173
Lee, R. C.-T., 358, 1129
Lee, T.-M., 1160
Leech, G., 937, 973, 1152, 1164
Lefkovitz, D., 211, 1152
LEGAL-ACTIONS, 100
Lehmann, D., 522, 1150
Leibniz, G. W., 7, 136, 263, 555, 735
Leimer, H., 602, 1151
Leipzig, 15
Leiserson, C. E., 1115, 1131
lema de elevación, 342, **344**
Lemmer, J. F., 606, 1147
Lenat, D. B., 413, 423, 808, 1132, 1152
lenguaje, **897**, 898-900
  Aborrece la sinonimia, 957
  análisis, **900**
  como acción, 937
  comunicación, 19, 897-977
  conocimiento, 22, 27
  de comportamiento, **1060**
  de consultas, **952**
  de programación Ada, 17
  de programación, 272
  formal, **899**, **897**
  generación, 900
  modelo, 645, 953, **968**, 970
    natural, 5, 273, **899**
  percepción, **900**
  robótico genérico, **1060**
  situado, 936
  traducción, 24, 791, 850-857
  y pensamiento, 275
Leniewski, S., 415
Lenstra, J. K., 99, 1151, 1173
lente, **983**
Lenzerini, M., 417, 1128
Leonard, H. S., 1152
Leonard, J. J., 1066, 1152
Leone, N., 1135, 1140
Lesh, N., 1136
Lésniewski, S., 1152
Lespérance, Y., 1060, 1139, 1152
Lesser, V. R., 522, 659, 1135
Lettvin, J. Y., 1013, 1152
Letz, R., 358, 1152
Levesque, H. J., 415, 417, 419, 522, 1060, 1126 ,1130, 1139, 1152, 1168
Levitt, G. M., 208, 1152
Levitt, T. S., 1067, 1150
Levy, D. N. L., 212, 1152
Lewis, D. D., 974, 1152
Lewis, D. K., 63, 938, 1097, 1152
Lewis, R. A., 1148
LEX, 768, 782
Léxica y funcional

lexicalizado PCFG, 949, 973
Ley de Murphy, 97, 104
Ley Moore de, 1094
ley social, **515**
leyes del pensamiento, 4
LFG, *véase* gramática
Li, C. M., 265, 1152
Li, M., 768, 1152
liberar espacio, **1042**
Libre albedrío, 7, 1084
Lieberman, L., 1017, 1123
LIFE, 357
Lifschitz, V., 417, 418, 466, 1139, 1152
Lighthill, J., 25, 1052
Límite de profundidad, 194
limpio vs. desaliñado, 29
Lin, D., 861, 1127
Lin, F., 414, 1052
Lin, S., 99, 146, 1053
Linden, T. A., 521, 1053
Lindsay, R. K., 413, 975, 1053
línea
  base, **997**
  de búsqueda, 137
  de retardo, 257
lineal
  álgebra, 1112-1113
  Gaussiana, **572**, 602, 628
  programación, 149, 726
  regresión, **820**
  resolución, **346**, 803
    de entrada, 350
  restricción, **157**
  separable, **843**, 858
  separador, **843**, 853
    óptimo, 853
  suavizado de interpolación, **947**
linealización, **442**, **1035**
líneas paralelas, 982
lingüística, 18, 34
  computacional, 19
LINUS, 803
Lipkis, T. A., 417, 1167
Lisp, **22**, 280, 389, 763, 1061
LISTA, 316
  accion concurrente, **514**
  de decisión, **763**
  de enlace, **959**
  de ligaduras, **287**
listas, **292**
literal
  de respuesta, **341**
  negado, 350
  (sentencia), **229**
literales complementarios, 240
Littman, M. L., 33, 149, 521, 735, 1121, 1128, 1147, 1153, 1154
Liu, J. S., 579, 1153
Liu, W., 1129
LLAMA, 330, 332
Lloyd, E. K., 1125
Lloyd, J. W., 356, 1153
localidad, 259, **596**
localización, **1031**
  global, **1031**
  y asociación simultáneas, **1036**
localizador uniforme de recursos (URL), 392
Locke, J., 7, 937, 1153
Locke, W. N., 975, 1153
Lodge, D., 1106, 1153
logaritmo de la verosimilitud, **816**
Logemann, G., 248, 1132
lógica, 4, 9, 224-229
  atómicos, 281
  borrosa (difusa), 530, 599-600, 605
  cuantificador, 281-286
  de orden superior, 274
  de Port-Royal, 664, 690
  de primer orden, **271**, 271-307
  descriptiva, 397, 401-402, 412, 417
  dinámica, 414
  igualdad, 286
  inductiva, 538, 556, **556**
  inferencia, 310-312
  interpretaciones, 278-280
  modal, **388**, 415
  modelos, 277-278
  muestreo, 603
  no monotónica, **407**
  notación, 4
  orden superior, **274**
  por defecto, **408**, 412, 417
  probabilista de primer, 590-595
  proposicional, 218, 229-236, 272
    inferencia, 233-251
    semántica, 230-232
    sintaxis, 229-230
  resolución, 239-244
  semántica, 277
  sintaxis, 277
  temporal, **274**, 414
  términos, 280, 331
lógico
  agente, 219
  conectiva, 19, 229, 281
  inferencia, 227, 310-362
  omnisciencia, **389**
  positivismo, **7**
  razonamiento, 236-253
logista, 5
Logistello, 204
Lohn, J. D., 149, 1153
Londres, 17
Long, D., 468, 1137
LONGITUD, 134, 454, 732, 910, 1109
  de onda 985
Longuet-Higgins, H. C., 1153
Lotem, A., 1173
lotería, **666**, 693
  estándar, **673**
Lovejoy, W. S., 645, 1153
Lovelace, A., 17
Loveland, D., 248, 356-358, 1132, 1153
Lowe, D. G., 1017, 1153
Löwenheim, L., 303, 1153
Lowerre, B. T., 148, 658, 1153
Lowrance, J. D., 522, 1153, 1175
Lowry, M., 1142
Lowry, M. R., 358, 1153
Loyd, S., 99, 1153
Lozano-Pérez, T., 1067, 1127, 1153, 1174
LPG, 468
LRTA*, 144, 145, 150, 151
LRTA*-AGENT, 144, **144**
LRTA*-COST, 144
LSTD, 892
Luby, M., 129, 603, 658, 1131, 1153
Lucas, J. R., 1078, 1079, 1153
Lucas, P., 606, 687, 1153
Luce, D. R., 11, 1153
Luger, G. F., 35, 1154
Lugosi, G., 859, 1134
Lull, R., 7
LUNAR, 938
Luong, Q.-T., 1136
Lusk, E., 1176
luz, 983-985

MA*, *véase* búsqueda, limitada por memoria A*
MacHack 7, 210
Machover, M., 304, 1124
MacKay, D. J. C., 861, 1154, 1156
Mackworth, A. K., 165, 177, 178, 1017, 1138, 1154, 1163

macrop (macro operador), **519**
Madigan, C. E, 1158
Mahanti, A., 151, 1154
Mahaviracarya, 555
Maier, D., 356, 1123, 1124
Majercik, S. M., 521, 1154
Makov, U. E., 1172
maleabilidad de la inferencia, 401
Mali, A. D., 1147
Malik, J., 990, 993, 998, 1017, 1124, 1133, 1144 ,1154, 1168
Malik, S., 1158
Mandela, N., 1004
Manhattan, *véase* heurística, Manhattan
manipulador, **1023**
Mann, W. C., 940, 1154
Manna, Z., 304, 357, 358, 1154
Manning, C. D., 974, 975, 1148, 1154
Manolios, P., 1147
manos de póker, 557
Mansour, Y., 1147, 1171MAP (máximo a posteriori), 814
mapa
de reflectancia, **999**
de rutas probabilísticas, **1047**
Máquina
Abstracta de Warren, 331, 356
Analítica, 17
de Boltzmann, **863**
de estado, 901
de estados finitos aumentada, **1058**
de Turing, 10, 768
mano-ojo, **1067**
núcleo, **852**, 851-855
ultrainteligente, 1094
Marais, H., 1169
Marbach, P., 893, 1154
marcadores, **1031**
Marcinkiewicz, M. A., 1154
marco de coordenadas, 992
Marcu, D., 1139
Marcus, M. P., 973, 1154
marginal, 546
marginalización, **542**
Marín, J., 768, 1145
Markov, **1067**
cadena Monte Carlo, **587**, 587-590, 601, 603, 641
cadena, **589**, 613
manto, **569**, 610
proceso de decisión (MDP), 11, **699**, 732, 734, 868
parcialmente observable (PDMPO), 712, 713, 712-715, 734
proceso, **613**
propiedad, 622, 656, 699
Markov, A. A., 656, 972, 1154
Maron, M. E., 556, 973, 1154
Marr, D., 1154
Marriott, K., 177, 178, 357, 1154
Marsland, A. T., 212, 1154
Martelli, A., 147, 357, 1154
Marthi, B., 658, 1154
Martin, J. H., 34, 658, 940, 941, 975, 1146, 1154
Martin, N., 355, 1127
Martin, P., 519, 940, 1143, 1154
más restringida, 322-323
Maslov, S.Y., 354, 1154, 1155
Mason, M., 100, 520, 1069, 1135, 1153, 1155
Mataric, M. J., 1068, 1155
Mateis, C., 417, 1135
matemática, 9-11, 21, 33
materia, 372
materialismo, 7, **1084**, 1097
eliminativo, **1097**
Mates, B., 263, 1155
Matheson, J. E., 679, 692, 1144, 1156
matriz, **1112**
de ganancia de Kalman, **632**
de transiciones, 610
Maturana, H. R., 1152
Mauchly, J., 16
MAX, 186, 190
máxima
utilidad esperada, **664**, 668, 690
verosimilitud, **815**, 815-820
maximin, **723**
máximo a posteriori, **814**
MAX-VALOR, 186, **186**, 190, **190**
Maxwell, J., 913, 938, 1016, 1155
Mayer, A., 148, 1141, 1142
Mayne, D. Q., 658, 1141
MAYORÍA-PONDERADA, 759
Mazumder, P., 99, 1168
MCA, 164, 168, 177
McAllester, D. A., 30, 211, 351, 418, 467, 938, 1147, 1155, 1171
MCC, 28
McCarthy, J., 20, 21, 64, 262, 303, 354, 388, 414, 417, 1089, 1155
McCartney, R. D., 358, 1153
McCawley, J. D., 938, 1155
McClelland, J. L., 29, 1166
McConnell-Ginet, S., 34, 938, 1129
McCulloch, W. S., 18, 19, 23, 265, 838, 840, 860, 1152, 1155
McCune, W., 348, 351, 358, 1155
McDermott, D., 2, 355, 398, 417, 468, 520, 521, 1129, 1139, 1155
McDermott, J., 28, 325, 355, 1155
McDonald, R., 273
McEliece, **R. J.**, 604, 1155
McGill, M. J., 859, 973, 1167
McGregor, J. J., 177, 1156
McGuinness, D. L., 1126
McKeown, K., 939, 1156
McLachlan, G. J., 859, 1156
MCMC, *véase* cadena de Markov Monte Carlo
McMillan, K. L., 469, 1156
McNealy, S., 1092
MDL, *véase* longitud de la descripción mínima
MDP, *véase* proceso de decisión de Markov
mecánica de la estadística, 29, 861
mecanismo, **729**
medición de inteligencia, 23, 35
medida, **369**
del rendimiento, 40, 44, 585
medidas, 369-371
medio de limpieza, 41
Meehan, J., 355, 1129
Meehl, P., 1077, 1141, 1156
mega, 624
Meggido, N., 736, 1149
mejor
hipótesis actual, 768, 776
premio posible, 673
MEJOR-APRENDIZAJE-ACTUAL, 778
Meléuk, I. A., 938, 1156
Mellish, C. S., 357, 1130
memoria, 275
a corto plazo, 325
asociativa, **862**
de largo término, 325
de trabajo, 807
memorización, 334, 353, **787**
MEMS, 1100
Mendel, G., 135, 1156
meningitis, 547, 559
mente, 2, 1094
ciencia, 4
como sistema físico, 7
filosofía de, 1096, 1098
visión dualista, 1096

y misticismo, 14
Mercer, J., 751, 1156
Mercer, R. L., 973, 1127, 1146
Merkhofer, M. M., 691, 1156
MERLIN, 416
Merlin, P. M., 356, 1129
metadatos, 957
Meta-DENDRAL, 768, 782
metafísica, 7
metáfora, **929**, 939
meta (objetivo), **57**, **68**, **87**, **98**
agente basado en, 57-58, 61
formulación, **68**, 492
predicado, **744**
razonamiento dirigido, 207, 247
serializable, **464**
test, **70**, **98**, 156, 436, 907
meta-razonamiento, **207**
teórico de decisión, **1103**
meta regla, **335**
método
camino critico, 476
de alineación, 1010, 1010-1012
de conexión, 354
débil, 26
metonimia, **928**, 939
Metrópolis, N., 149, 603, 1156
MEU, *véase* máxima utilidad esperada
Mézard, M., 862, 1156
MGONZ, 1077
MGSS*, 211
Michalski, R. S., 768, 769, 1156
Michaylov, S., 356, 1145
Michel, S., 861, 1156
Michie, D., 34, 77, 146, 147, 211, 520, 769, 886, 892, 1067, 1134, 1156
micromortalidad, **673**, **691**
micromundo, 23, 25
MICRO-PLANNER, 355
Microsoft, 602, 606, 961
Middleton, B., 1163
miembro (en una escena), **1001**
Milgrom, P., 736, 1156
Mill, J. S., 8, 776, 1156
Miller, A. C., 691, 1156
Miller, D., 1133
Millstein, T., 1136
Milner, A. J., 1140
MIN, 186, 190
MIN-CONFLICTS, 170, **170**
MINIMAX, **190**
MINIMAX-DECISIÓN, **186**, 192
Mínimo
compromiso, **441**, 779
intervalo de relajación, **482**
mínimos valores restantes, 162
Minker, J., 355, 1138
Minsky, M. L., 19, 22, 26, 28, 34, 416, 529, 757, 1155, 1156
Minton, S., 148, 177, 519, 806, 1128, 1157
MIN-VALOR, 186, **186**, 190, **190**
Mira mamá, sin manos, 21
misioneros y caníbales, 98, 101, 412
misticismo, 14
MIT, 20-22, 1067
Mitchell, D., 149, 265, 1131, 1168
Mitchell, M., 149, 1157
Mitchell, T. M., 64, 768, 769, 782, 806, 1102, 1127, 1156, 1157
Mitkov, R., 939, 1157
mixtura
de Gausianas, 648, **827**, 829
distribución, 826
ML, *véase* lógica modal de probabilidad máxima, 416
modelo
acústico, **645**
al deshacer ambigüedad, 930
bigram, 654, 652
de base de datos, **123**, 148
disjunta, **124**
de error Gausiano, **637**
de espacio vectorial, **960**, 974
de fallo, 639
de fallos persistente, **639**
de Markov oculto, 30, 611, **624**, 624-627, 635, 656, 657, 832
de movimiento, **1030**
de observación, 614, **713**
de probabilidad relacional, **591**
de pronunciación, 649
de traducción, **968**
de transferencia (in MT), **966**
de transiciones, 614, 643, 656, **699**, 732
del mundo, en eliminación de ambigüedad, 930
del vecino más cercano, 835-838
estable, **406**, 405-406
gráfico, 562
kernel, 837, 837-838
mental, en desambiguación, 930
mínimo, 404, **407**
modelos de probabilidad temporal, 593
módulo de decisión, **759**
Modus Ponens, **237**, 263, 347, 352, 360, 389
Moffat, A., 1175
Mohr, R., 165, 177, 1157
Mohri, M., 901, 1157
Moisl, H., 940, 1131
MOM, 624, 635, 659
Monis, P., 1146
monismo, **1084**
monitorización, **616**
de ejecución, 491, 502-507, 518
monos, 897
de Vervet ,897
y plátanos, 101, 470
monotonicidad
(de preferencias), **667**
(de una heurística), **113**
monótono, **238**, 407
Montague, P. R., 892, 1157, 1167
Montague, R., 415, 416, 938, 1147, 1157
Montanari, U., 147, 176, 357, 1125, 1154, 1157
Monte Carlo localización, **1033**
Montemerlo, M., 1067, 1157
montón, 369
Mooney, R., 807, 940, 1134, 1157
Mooney, R. J., 934, 940, 1157, 1176
Moore, A. W., 148, 1126, 1147, 1157
Moore, E. F., 100, 1157
Moore, J. D., 939, 1123, 1176
Moore, J. S., 351, 358, 416, 1126, 1147, 1157
Moore, R. C., 416, 419, 1143, 1157
Moravec, H. P., 1066, 1086, 1094, 1157, 1158
More, T., 20
Morgan, J., 522, 937, 1130
Morgan, N., 659, 1140
Morgenstern, L., 416, 418, 1158
Morgenstern, O., 11, 208, 692, 1173
Moricz, M., 1169
Morjaria, M. A., 606, 1158
morphing, 1008
Morra de dos dedos, 719 TIPO, 903
Morris de nueve-hombres, 204
Morrison, E., 208, 1158
Morrison, P., 208, 1158
Moses, Y., 416, 1136
Moskewicz, M. W., 265, 1158
Mosteller, F., 861, 1158
Mostow, J., 148, 152, 1158

motor diferencial, 17
Motzkin, T. S., 861, 1158
Moussouris, J., 210, 1158
Moutarlier, P., 1066, 1158
movimiento
aparente, 993
obediente, **1039**
punto a punto, **1039**
Mozetic, 1, 768, 1156
MPI, *véase* independencia preferencial mutua
MRS (sistema de razonamiento metanivel), 335
MUESTRA-PONDERADA WEIGHTED SAMPLE, 585, 586
POR-PRIORI, 582, 583, 584
Muestreador de Gibbs, **592**, 603
muestreo, 582-587
de importancia, 603
por rechazo, **583**
MUESTREO-RECHAZO, 583, **584**
Muggleton, S. H., 803-805, 808, 940, 1135, 1148, 1158, 1170, 1172
Müller, M., 212, 1158
Muller, U., 862, 1018, 1152
múltiple, **399**
Mundo
cerrado, 403-405
de bloques, 23, 27, 418, 434-436, 508
de la aspiradora, 39, 65, 104, 493, 499, 895
posible , **225**, 302, 416, 591
wumpus , **221**, 221-224, 233, 266, 292-295, 365, 550-554, 560, 898
Mundy, J., 1158
MUNIN, 602
Murakami, T., 204
murciélagos, 522
Murga, R., 1151
Murphy, K., 604, 657, 1067, 1125, 1158
Murphy, R., 1069, 1150, 1158
Muscettola, N., 64, 519, 1146, 1158
música, 14
mutación, 25, **133**
MUTAR, 134
mutex, **396**
MYCIN, 27, 598, 605
Myers, K. L., 1175
Myerson, R. B., 735, 1158
Myrhaug, B., 416, 1125, 1131
Nadal, J.-P., 862, 1156
Nagel, T., 1096, 1158
Nalwa, V. S., 14, 1018, 1159
Nardi, D., 417, 1128
NASA, 32, 355, 418, 519, 606, 1024
Nash, J., 1159
NASL, 521
naturalismo biológico, **1083**
Nau, D. S., 211, 519, 1136, 1150, 1159, 1170, 1173
Naur, P., 937, 1159
navaja de Ockham, **742**, 766-769, 783, 801, 814
Navidad, 1082
NAVLAB, 32, 1024
Nayak, P., 418, 519, 1158, 1159
Neal, R., 861, 1159
Nealy, R., 203
Nebel, B., 466, 468, 1149, 1159
negación, **229**
como fallo, 329, **405**, 405-406
negativo falso, **775**
negligencia, 1092
Nelson, G., 357, 1159
NETL, 417
Netto, E., 99, 1159
neurobiología, 1018
neurociencia, 12-14, **838**
computacional, 832
NEUROGAMMON, 211, 885
neurona, **12**, **19**, 1087
neutralidad frente a riesgos, **671**
Nevill-Manning, C. G., 934, 940, 1159
Nevins, A. J., 355, 1159
NEW-, 330
Newborn, M. M., 1161
Newell, A., 4, 20, 21, 31, 64, 99, 146, 209, 263, 264, 355, 416, 455, 519, 807, 1103, 1151, 1156, 1158, 1168, 1170
Newman, P., 1066, 1134
Newton-método Raphson, 137
Newton, I., 1, 52, 136, 137, 148, 616, 1159
Ng, A. Y., 734, 889, 894, 1125, 1147, 1159
Nguyen, X., 468,1159
Niblett, T., 808, 1130
Nicholson, A., 658, 734, 1133, 1159
Nielsen, P. E., 355, 1146
Niemel, I., 1159
Niemel, L, 417
nieve, palabras para, 275
Nigenda, R. S., 1159
Nilsson, D., 691, 1160
Nilsson, N. J., 2, 31, 34, 64, 100, 146, 152, 209, 263, 303, 304, 341, 358, 466, 519, 604, 806, 861, 1066, 1072, 1136, 1139, 1142, 1160, 1174
Niranjan, M., 1132
NIST, 855
nivel
de implementación, **220**
(en grafos de planificación), **450**
Nixon, R., 407
NLP (procesamiento de lenguaje natural), 3, 897-977
no
computabilidad, 9
determinista, *ver*
episódico, 48
holonómico, **1028**
monotonicidad, **407**
monotónico lógico, 407-409, 418
NOAH, 467, 520
Noda, I., 1066, 1148
NODO, 81
búsqueda, **78**
de decisión, 679
de oportunidad (red de decisión),**679**
de posibilidad (árbol de juego), 197
determinista, **570**
hoja, **80**
padre, **78**
valor, *véase* utilidad valor nodo, computación, 1103
nombres compuestos largos, 299
nomenclatura binomial, 413
NONLIN, 467
NONLIN+, 518
Nono, 319
norma del máximo, **707**
normalización, **543**
NORMALIZAR, 544, 577, 579, 586, 588
Norman, D. A., 975, 1125
North, T., 26, 1138
Norvig, P., 31, 355, 371, 569, 658, 939, 1147, 1160, 1166
NO-SOLUCIÓN-POSIBLE, 454
notación
aritmética, 5
lógica, 4
Nourbakhsh, I., 100, 1139

Nowatzyk, A., 1144
Nowick, S. M., 352, 1160
Nowlan, S. J., 149, 1143
NP (Frase Nominal), 900
NP (problemas duros), 1110-1112
NP-completo, 10, 74, 99, 157, 236, 238, 264, 417, 580, 916, 1004, 1017, 1111, **1111**, 1115
NSS programa de ajedrez, 209
NUEVA CLÁUSULA, **800**
NUEVOS LITERALES, 800, 801
número de
  conspiración, 210
  teléfono, 390
números naturales, **290**
Nunberg, G., 939, 1160
Nussbaum, M. C., 8, 1096, 1160
Nygaard, K., 416, 1131

O( ) notación, 1110
o bjetivismo, 538
OBJETIVOS, 454
objetivos serializables, **464**
objeto, 280
  compuesto, **368**
objetos, 273
  mentales, 387-391
O'Brien, S., 974, 1163
observable, 47
Ockham, W., 742, 767
octante, **1002**
oculta, **574**
odometría, **1026**
o exclusiva, 232, 865
Ogasawara, G., 657, 1144
Ogawa, S., 13, 1160
Oglesby, F., 1141
ojo por ojo, **727**
ojos, 979, 983, 985, 1015
  de mosca, 1013
Olalainty, B., 1138
Olawsky, D., 520, 1160
Olesen, K. G., 602, 603, 1122, 1160
Oliver, R. M., 692, 1160
Olshen, R. A., 768, 1127
Olson, C. F., 1018, 1160
Olum, P., 1016, 1139
omnisciencia, **42**
  lógica, 389
Omohundro, S., 973, 1171
Oncina, J., 940, 1128, 1160
ONTIC, 351
ontología, **296**, 299
  general, 363-366, 412
  superior, **364**, 412
OP, 316
opacidad referencial , 416
opaco, 388
operacionalidad, 790, **790**
operador modal, **388**, 414 modelo, 56, **225**, 302, 591
O-PLAN, 489, 518, 519
Oppacher, F., 149, 1160
Oppen, D. C., 357, 1159
OPS-5, 325, 355
óptima de Pareto, **721**
optimalidad
  (de un algoritmo de búsqueda) 80, 98
  limitada, **1105**
    asintótica (ABO), **1105**
optimizador, agujero, 488
OPTIMUM-AIV, 519
ORDEN, 483
ordenación, **666**
  de conjuntos, **322**
O'Reilly, U.-M., 149, 1160
Organon, 263, 413
orientación, **992**
orientada a objetos programación, 18, 399
Ormoneit, D., 893, 1160
oro, 221
O'Rorke, P., 808, 1175
Ortony, A., 939, 1160
Osawa, E., 1066, 1148
Osborne, M. J., 735, 1160
Oscar, 522
óscar de la academia, 522
Osherson, D. N., 768, 1160
Ostland, M., 657, 1161
Otelo, 182, 204
OTTER, 349, **349**, 351, 358, 361
Overbeek, R., 358, 1176
Overmars, M., 1068, 1147
Owens, A. J., 1137
oyente, **898**

P (vector probabilidad), 535
PAC, *véase* probable y aproximadamente correcto
Page, C. D., 808, 1130, 1160
Page, L., 974, 1130
PAGERANK, 974
Pak, I., 659, 1160
palabra, **898**
  de terminación, **846**
Palay, A. J., 210, 1160
Pallottino, S., 99, 1138
Palmer, D. A., 975, 1160
Palmer, S., 1018, 1160
Palmieri, G., 861, 1138
palomas, 15
Pang, C.-Y., 99, 1134
Panini, 18, 937
Papadimitriou, C. H., 148, 150, 265, 734, 973, 1017, 1115, 1134, 1148, 1150, 1160, 1161
Papadopoulo, T., 1016, 1136
Papavassiliou, V., 893, 1161
Papen, S., 26, 861, 1156
PARADISE, 207
paradoja, 416, 690, 693
  de San Petersburgo, 690, 693
paralaje de movimiento, 995, 1015
paralelismo
  O-, **332**
  Y-, **332**
parámetro, **571**, 815
  de ganancia, **1052**
paramodulación, **346**, 357
Pardalos, P. M., 1135
Parekh, R.. 940, 1161
PARENT-NODE, 78, 79, 81
Parisi, D., 940, 1135
Parisi, G., 604, 1161
Park, S., 351, 1142
Parker, D. B., 861, 1160
Parker, L. E., 1068, 1161
Parr, R., 894, 1105, 1161, 1166
Parrod, Y., 1121
PARTE, 903
partes meteorológicos, 964
partición, 367
Parzen, E., 756, 1161
Pascal, B., 7, 10, 555
Pasero, R., 356, 1130
pasos condicionales, **494**
Pasula, H., 605, 657, 659, 1154, 1161
Paterson, M. S., 357, 1161
Patil, R., 417, 913, 938, 1129, 1135
pato, mecánico, 1065
Patrick, B. G., 147, 1161
Pattern. T., 939, 1161
Paul, R. P., 1068, 1161
Paull, M., 265, 1137
Pazzani, M., 556, 859, 1134
PCFG, *véase* gramática, de contexto libre
PDDL, 432, 466
PDP, *véase* procesamiento paralelo distribuido

Peano, G., 303, 1161
Pearl, J., 31, 64, 108, 146-148, 178, 209, 211, 559, 569, 602-606, 695, 859-860, 860, 1133, 1139, 1148, 1161, 1168
Pearson, J., 178, 1161
pecados, siete mortales, 127
Pecheur, C., 351, 1142
PEDIR-ACCIÓN, 219
Pednault, E. P. D., 466, 522, 1161
PEGASUS, 889, 890, 893, 896
Peirce, C. S., 176, 303, 397, 416, 937, 938, 1162
Peled, D., 358, 1130
películas
  2001: Una Odisea Espacial, 602
  A.I., 1095
  Attack of the Cat People, 928
  Coge el Dinero y Corre, 652
  The Matrix, 1093
  Terminator, 1093
Pelikan, M., 149, 1162
Pell, B., 63, 519, 1158
Pemberton, J. C., 151, 1162
Penberthy, J. S., 467, 1162
péndulo invertido, **886**
Peng, J., 892, 1162
Pengi, 521
penguin, 522
Penix, J., 351, 1142
PENMAN, 939
Penrose, R., 1078, 1162
pensamiento racional , 4
Pensilvania, U. de, 16
peor catástrofe posible, 673
Peot, M., 521, 603, 1162, 1168
percepción, 36, 38, 293, 979-1015
  tricromática, 986
perceptrón, 23, **842**, 858, 861
  votado, 861
Pereira, F., 328, 901, 937, 940, 1157, 1162, 1174
Pereira, L. M., 1174
Peres, Y., 580, 1154
pereza, **529**
perfil de estrategia, **720**
Perl, 389
perplejidad, **947**
Perrault, C. R., 937, 1130
Person, C., 892, 1157
perspectiva, 1016
pesado dinámico, 146
peso del sesgo, 839
Peters, S., 938, 1134
Peterson, C., 604, 1162
Petrie, T., 657, 860, 1123
Pfeffer, A., 605, 735, 1149, 1162
Pfeifer, G., 417, 1135
Philips, A. B., 1157
Picasso, P., 1082
Pickwick, Mr., 1082
Pijls, W., 1162
pila, 85
piloto automatizado, 357
ping-pong, 36
Pinker, S., 273, 940, 1162
Pisa, torre de, 61
pista, **331**
Pitts, W., 18, 19, 265, 838, 840, 860, 1155
Pitts, W. H., 1152
pixel, **981**
Plaat, A., 211, 1162
Place, U. T., 1096, 1162
Plamondon, P., 975, 1156
plan
  biblioteca, 482
  condicional, 518
  conjunto, **513**
  consistente, **444**
  continuo, 508
  defecto de, **510**
  no lineal, 467
  reconocimiento, 516
  replanificación, 503, 504
  universal, 521
PLAN-ERS1, 519
PLANEX, 521
planificación, 58, 207, **374**, **428**, 427-525
  basada en casos, **520**
  clásica, 428, 490, 492
  como satisfabilidad, 458-463
  condicional, 490-502, **491**, **517,** 528, 686
  conformista, 491, 518, 520
  contingente, 491
  continua, 491, 492, 507-512, 518, 521
  de fábrica de cervezas, 521
  del suelo, **179**
  de portaaviones, 521
  de primer orden, **442**
  de trayectorias, **1040**
  espacio de estados, 436-441
  gráfo, **450**, 450-458, 465
    serial, **453**
  heurística, 449-450
  historia de, 466
  jerárquica, 481-490, 518
  lineal, **467**
  movimiento certero, 1049
  multiagente, **512**, 512-517
  multicuerpo, **514**, 514-515
  mundo de bloques, 24
  no-intercalador, 471
  ordenada parcialmente, **442**, 441-450
  progresión, **436**
  reactiva, **521**
  red jerárquica de tareas, 481
  regresión, **438**, 467
  sin sensores, 491
  y actuación, 490-493
  y ejecución, 507-512
PLANIFICADOR, 28, 355, 503, 505, 903
  de regresión, *véase* planificación
  sin sensores, 491
Plankalkül, 16
Platón, 263, 415, 1095
Plotkin, G., 357, 807, 1162
Plunkett, K., 940, 1135
Pnueli, A., 414, 1162
poda, **114**, 182, 754
  de inutilidad, **203**
  en EBL, 790
  hacia delante, 196
  inutilidad, **203**
Poda χ^2, **755**
Podelski, A., 357, 1121
Poesía, 1
Pohl, I., 100, 146, 1162
Polguere, A., 938, 1156
poliárbol, **580**, 601, 621
Política, 522, **699**, 733
  apropiada, **703**
  búsqueda, **887**, 887-890
  iteración, **710**, 710-711, 733
    asíncrona, 711
  miope, **686**
  modificada, **711**
  óptima, **700**
  pérdida, **709**
Pollack, M. E., 468, 519, 522, 937, 1130, 1146, 1176
Polonia, 415
Pomerleau, D. A., 1016, 1163
PONDERACIÓN DE LA VEROSIMILITUD, 585, **586**, 601, 603, 642
Ponte, J. M., 1163
Poole, D., 2, 31, 63, 602, 605, 1163, 1177

POP, 442, 445, 448, 449, 465, 467, 484, 486, 488, 510, 511, 514, 515
-CON, 507
Popper, K. R., 555, 768, 1163
porbabilidad posterior, *véase* probabilidad, campo potencial condicional, **1045**
Porphyry, 416
Porter, M. F., 974, 1163
Portugués, 784
Posegga, J., 358, 1124
posibilidad de ganar, 192
positivismo, lógico, **8**
Post, E. L., 264, 1163
potencia de representación,25, 865
Prade, H., 605, 1135
Pradhan, M., 1163
pragmáticos, 899
PRAGMATICS, 903
Pratt, V. R.. 414, 1163
Prawitz, D., 354, 1163
precio, 422
precisión, **956**
precondición, 430
externa, 482
precondiciones abiertas, **444**, 511
predecesor, **90**, 438
PREDECIR, 910, 943, 909
predicado, 920
predicción, 619, 656
Predicciones de Simon, 24
preferencia, **531**, 666, 939
estacionaria, **702**
monótona, **669**
unitaria, 346
PREGUNTAR-MCMC, **588**, 589
premio
Loebner, 35
Nobel, 12, 26
premisa, 229
presidente, 386
Press, W. H., 149, 1163
Prete, F., 896, 1128
Price Waterhouse, 518
Price, B., 734, 1126
Prieditis, A. E., 122, 148, 152, 1158, 1163
prima del seguro, **671**
Princeton, 19
Principia Mathematica, 20
Prinz, D. G., 209, 1163
Prior, A. N., 414, 1163
prisioneros, tres, 559
probabilidad, 10, 31, 527-539, 561-610, 616, 1114-1115
alternativas, 595
a priori, 530, **534**, 534-537, 554
axiomas de, 537-541
condicional, 530, **536**, 536-537, 554, 565
conjunta, 565
de clase, **770**
de transición, **589**
evaluación, 538
fallos, 568
función densidad, **535**, 1114
historia, 556
marginal, 542
principio de teoría, 276, **529**, 690
probabilística y aproximadamente correcto, **761**, 764, 769
probabílistico, 534-537
Probador de Teorema de Tecnología Prolog (PTTP), 349, 358
problema, **70**, **97**
8-puzle, 119, 121, 146
8-reinas, 74, 99
acoplamiento funcional (procedimental), **395**, 400
bandido, 892
contingencia, **96**
de carro y mástil, **886**
de cualificación, 491, 528, 1079, 1081
de decisión, 10
de exploración, 138
de la parada, 312
de las 8 reinas, 75, 99
de los bandidos, 877
de ramificación, **378**
de rutas, **76**
de secuestro, **1031**
del mapa de cuatro colores, 176
del millón de reinas, 169
del mundo real **72**
del viajante de comercio, 76, 99, 147, 151
diseño VLSI, 77, 131
enfoque declarativo, **220**
evento, 415
formulación, **68**, **71-72**
generador, 61
inherentemente difícil, 1110-1112
juguete, **73**
mente-cuerpo, **1084**, 1084-1085
millones de reinas, 169, 177
misioneros y caníbales, 101
mono y plátanos, 101, 470
navegación de robot, 77
n-reinas, 251
parada, 312
relajado, 121, 122, 440, 449
resolución, 25
ruta, 76
secuencia de ensamblaje, 77
semidecidible , 312
ubicación de aeropuertos, 694
viajante de comercio, 76
problemas horizonte infinito, 733
procesamiento
de señal, **647**
paralelo distribuido, *véase* red neuronal
PROCESO, 349, **349**
proceso, **383**
de decisión compuesta, 147
estacionario, **613**, 613-616
procesos, 383-384
estacionarios, 613
mentales, 1083
PRODIGY, 519
producción, 53
PRODUCTO PUNTO A PUNTO, 579
producto punto a punto, **578**
profesor, 741
PROFUNDIDAD, 78, 81, 87
Profundidad de campo, **983**
profundización iterativa, *véase* búsqueda, profundización iterativa
PROGOL, 796, 803, 808
programa, **478**
Programación
de actividades, 476, 476
dinámica, 147, 334, 621, 733
adaptativa, 871
no en serie, **603**
funcional, 357
genética, **149**
lógica, 303, 328-335, 356
en tablas, 334, **334**, 356
inductiva (ILP), **786**, 795-801, 806
restricción, **334**, 334-335, 356
por conjuntos de respuestas, **357**
Prolog, 28, **328**, 353, 467, 801, 937, 940
paralelismo, 322
promediado sobre clarividencia, 201
Pronunciación, **646**

propagación
cíclica, 604
unitaria, 249
propiedad
extrínseca, **373**
intrínseca, **372**
(relación unitaria), 274
proposición, 532
proposicionalización, 312
PROSPECTOR, 605
Prosser, P., 177, 1163
Protágoras, 937
prototipado rápido, 328
Proust, M., 967
Provan, G. M., 1163
PROVERBIO, 33
proyección, **374**
de perspectiva, **982**, 992
ortográfica ampliada, **983**
Proyecto
de Cadena Linguística, 938
de Programación Heurística (PPH), 27
de quinta Generación, 28
PRS (Sistema de Razonamiento Procedimental), 522
PRUEBA, 906
prueba, 238, **530**, **870**
de Turing, 3, 3-4, 5, 34, 35, 1077
comprobador, **351**
procedimiento, 9
Pryor, L., 521, 1163
pseudocódigo, 1118
pseudo-experiencia, 875
psicofísica, 1018
Psicología popular, 15, 419, 1097
psicología, 14-16
experimental, 4, 15
PSPACE, 1112
PSPACE-completo, 456, 466
PUCCINI, 521
Pueden pensar las máquinas, **1075**
puerta (lógica), 298
Puget, J.-F., 808, 1165
Pullum, G. K., 275, 901, 938, 940, 1144, 1163
PUNTERO-GLOBAL-PISTA, 330
punto
crítico, 252
de elección, **330**
de fuga, **982**
fijo, 245, 321
fotosensible, 1013
puntos maestrales , 534
Puterman, M. L., 63, 733, 1163
Putnam, H., 63, 248, 264, 341, 354, 556, 1096, 1132, 1163
Puzicha, J., 1018, 1124
puzle, 180
de bloque deslizante, **74**
8-puzle, 73, 99, 102, 119, 121, 146
Pylyshyn, Z. W., 1097, 1163

Q(a, s) (valor de la acción en estado), 880
QA3, 303, 414, 466
QLISP, 357
qualia, **1085**, 1087, 1097
Qubic, 204
reiterativo, **726**
robot (con humanos), 1072
suma cero, 215, **723**
teoría, 11, 697, **719**, 719-729
Quevedo, T., 208
Quillian, M. R., 416, 1163
química, 26
Quine, W. V., 304, 370, 413, 415, 1163
Quinlan, E., 975, 1163
Quinlan, J. R., 767, 771, 798, 801, 808, 1163, 1164
Quirk, R., 938, 1164

R1, 28, 325, 355
Rabani, Y., 149, 1160
Rabideau, G., 519, 1138
Rabiner, L. R., 659, 1164
Rabinovich, Y., 1164
racionalidad, 2, 40-42
calculadora, **1104**
limitada, 6, **1104**
perfecta, 6, 1104, **1104**
radar, 11
Radio Rex, 658
Raghavan, P., 973, 1160
raíces cuadradas, 52
Raiffa, H., 11, 672, 678, 691, 736, 1148, 1153
Rajan, K., 32, 63, 1146
Ramakrishnan, R., 355, 1165
ramificación, 433
Ramsey, F. P., 11, 555, 691, 1165
Rao, A., 64, 1176
Rao, B., 657, 1144
RAP, 522
Raphael, B., 146, 354, 807, 1125, 1141, 1142
Raphson, J., 137, 148, 1164
Raschke, U., 1130
Rassenti, S., 736, 1164
ratio de ganancia, **756**, 771
Ratner, D., 99, 1164
Rauch, H. E., 657, 1164
Rayner, M., 791, 1167
Rayson, P., 973, 1152
Razonador Socrático, **351**
razonadores automáticos, *véase* probadores de teoremas
razonamiento
analógico, 807
basado en
casos, 807
modelo, **295**
del mundo real, 23
incierto, 31, 902
intercausal, 598
lógico, 236-269
orientada a metas, 207
por defecto, 402-409, 595
espacial, **419**
intercausal
legal, 36
por defecto, 402-409
psicológico, **419**
temporal, 611-662
RBDTL, 794, 807
RBL, *véase* aprendizaje basado en relevancia
realimentación por relevancia, **957**
Reboh, R., 357, 1166
Rechenberg, L, 149, 1164
recompensa, 62, **699**, 732, 740, 868
media, 703
reconocimiento
basada en características, 1008-1010
basado en brillos, 1007
de dígitos, 855-857
de escritura, **1004**, 855-858, 1004, 1006
de objetos, **991**, 1004-1012
del habla, 30, 624, **645**, 645-656, 658, 900
óptico de caracteres, 900
recurso
consumible, 478
(en planificación), 478, 517
restricciones, 478-481
reutilizable, 478
recursos, 475-481
red
Bayesiana, 31, 561-569, 610

aprendizaje, 823-824
variables ocultas, 833-834
causal, *véase* red Bayesiana
conectada nodo a nodo, **580**
de alimentación positiva, **840**
de decisión, 562, 663, 679-681, 690, 692, 716
de tareas, 467
dinámica, **635**, 635-645, 656, 658, 699, 717, **716**, 733
evaluación, 681
híbrida, **571**
inferencia, 574-581
jerárquica de tareas (RJT), **481**
neuronal 19, 23, 29, 204, **838**, 838-851
aprendizaje, 19, 851
capa simple, *véase* perceptrón
expresividad, 19
función base radial, 862
hacia delante, **840**
interpretación probabilista de, 845
multicapa, 26, 846-850, 858
perceptrón, 842-846
probabilística cualitativa, 606, **676**
recurrente, **840**, 862
Reddy, R., 580, 1136, 1153
redes semánticas, 397-399, 412, 416
reducción, 1115
reducto, **406**
REESCRIBE-PARA, 910
reescritura de términos, 357
referentes, **925**
reflexión, 985
reflexión
difusa, 985
especular, **985**
refuerzo, 740, 868
refutación, **236**, 349, 354
Regalos de los Reyes Magos, 487
Regin, J., 177, 1164
regla
condición-acción, **53**, 212 por defecto, **406**
de Bayes, 11, 546-550, 554, 558, 645
de cadena, **565**
de Dempster, 598
de diagnóstico, **294**
del producto, **536**
delta, 883
esquema, **918**
implicación, 229
incierta, 596
punteadas, 908
si-entonces, 53, 230
situación-acción, 53
REGLA-
ACCION, 54, 56
CORRESPONDENCIA, 54, 56
reglas
causales, **294**
de Morgan , 285
de reescritura, 348, 361, **900**, 1117
recursivas por la izquierda, 906
regresión, lineal, **820**
Reichenbach, H., 556, 1164
Reif, J., 1068, 1128, 1164
reificación, **388**
REINFORCE, 889, 893, 896
Reingold, E. M., 176, 1125
REINICIALIZA-PISTA, 330, 331
Reinicio aleatorio, 153
Reino Unido, 28
Reiss, M., 205
Reiter, E., 939, 1164
Reiter, R., 265, 414, 417, 734, 1060, 1126, 1152, 1164
Reitman, W., 212, 1164
rejilla, rectangular, **92**
relación, 273
relaciones coherentes, **932**
relajación, **477**
relevancia, 232, 438, 785, 807
remuestreo, 658
Remus, H., 211, 1164
renombramiento, **320**
Rényi, A., 555, 1164
replanificación, **491**, 492, 502-507, 520
REPOP, 468
representación del conocimiento incierto, 461-464
REPRODUCIR, 134, **134**
requisitos de memoria, 84, 87
Rescher, N., 414, 1164
resolución, 23, 25, **240**, 239-244, 303, 335-353, 809
binaria, **338**
cierre, **243**, 344
de entrada, **347**
de juegos, 182-187
de referencia, **931**
estrategias de, 346-347
inversa, **801**, 801-805, 808
lineal, **347**
prueba de completitud para, 341
teórica, 358
unitaria, **240**
Resolvedor General de Problemas, 4, 466
resolvente, 801
responsabilidad, 1092
respuesta, 15
de sistemas, 1089
REST, 316
restricción
binaria, 159
de derivación, 783, 797, 805
de desigualdad, **449**
especial, 166
grabación, 177
grafo, 156, 173
no lineal, 158
orden, 447
preferencia, 159
propagación, 164, 163-166
recurso, 167
restricción, 797
unario, 158
restricciones
de estados, 461
de integridad, **245**
de unión, **1040**
ordenadas, **443**
RESUELVE-LP, **242**, 242, 243
RESULTADO, 100
rete, **325**
retina, 979
RETRACTARSE, 409, 410
retro-alimentación (propagación hacia atrás), 26, 29, 848, 846-851, 858, 861
Reversi, 204
revolución, **1027**
Reynolds, C. W., 522, 1164
Ribeiro-Neto, B., 974, 1123
Rice, T. R., 1156
Rich, E., 2, 1164
Richardson, M., 658, 1165
Richardson, S., 603, 1139
Rieger, C., 28, 939, 1165
Riesbeck, C., 28, 355, 938, 1133, 1167
Riley, M., 901, 1157
Ringle, M., 1097, 1165
Rink, F. J., 606, 1158
Rintanen, J., 520, 1165
Ripley, B. D., 863, 1031
Rissanen, J., 768, 1165
Rissland, E. L., 34, 1170

Ristad, E. S., 943, 1123
Ritchie, G. D., 809, 1165
Ritov, Y., 657, 1161
Rivest, R., 769, 1115, 1131, 1165
Roberts, L. G., 1017, 1165
Roberts, M., 209, 1124
Robertson, N., 178, 1165
Robertson, S. E., 556, 974, 1165
Robinson, A., 354
Robinson, G., 357, 1176
Robinson, J. A., 22, 264, 303, 335, 341, 354, 1165
robot, 1066
  cognitivo, **415**
  fútbol, 65, 182, 522, **1064**
  humanoide, **1024**
  juego (con humanos), 1072
  móvil, **1024**
  navegación, 77
  robótica, **3**, 637
Roche, E., 974, 1165
Rochester, N., 20, 21, 1155
Rock, I., 1018, 1165
Rockefeller Foundation, 975
Roossin, P., 1127
ropas de limpieza, 942
Rorty, R., 1097, 1165
Rosenblatt, E., 844, 861, 1016, 1139, 1165
Rosenblatt, M., 860, 1165
Rosenblitt, D., 467, 1155
Rosenbloom, P. S., 31, 355, 519, 807, 1151
Rosenblueth, A., 18, 1165
Rosenbluth, A., 1156
Rosenbluth, M., 1156
Rosenfeld, E., 35, 1122
Rosenholtz, R., 1017, 1154
Rosenschein, J. S., 736, 1165
Rosenschein, S. J., 64, 265, 1068, 1147, 1165
Rosenthal, D. M., 1097, 1165
Ross, S. M., 556, 1165
Rossi, F., 1125
rotación, 992
Roussel, P., 303, 356, 1130, 1165
Rouveirol, C., 808, 1165
Roveri, M., 468, 1124, 1125, 1130
Rowat, P. F., 1067, 1165
Roweis, S. T., 657, 1165
Rozonoer, L., 863, 1121
RPM, *véase* modelo de probabilidad relacional
RSA (Rivest, Shamir, y Adelman), 351
Rubin, D., 658, 860, 863, 1134, 1139, 1165
Rubinstein, A., 735, 1160
ruido, **748**, 753-755, 782, 795, 811
ruido-OR, **570**
Rumanía, 68, 156
Rumelhart, D. E., 29, 861, 1165, 1166
ruptura, 1002
Rusia, 25, 209, 539, 656
Ruspini, E. H., 605, 1166
Russell, B., 7, 20, 353, 1175
Russell, J. G. B., 691, 1166
Russell, S. J., 31, 147, 211, 335, 371, 569, 657, 659, 734, 807, 860, 893, 894, 1061, 1067, 1103-1105, 1122, 1125, 1132, 1135, 1144, 1147, 1154, 1158, 1159, 1161, 1166, 1172, 1177
Rustagi, J. S., 603, 1166
ruta de referencia, **1052**
Ruzzo, W. L., 938, 1141
Ryder, J. L., 211, 1166

sabiendo que, 390
sabiendo si, 390
Sabin, D., 1166
Sacerdoti, E. D., 358, 467, 520, 1166
Sackinger, E., 862, 1018, 1151
Sacks, E., 418, 1166
Sadri, F., 415, 1166
Sag, I., 938, 940, 1146, 1166
Sagalowicz, D., 357, 1166
Sager, N., 937, 1166
Sagiv, Y., 356, 1123
Sahami, M., 606, 974, 1149, 1167
Sahni, S., 99, 1144
SAINT, 23
Salisbury, J., 1068, 1155
Salmond, D. J., 657, 1140
Salomaa, A., 973, 1167
Salto
  atrás cronológico, 168
  atrás dirigido por conflicto, **168**
  hacia atrás (backjumping), 177
Salton, G., 859, 973, 1167
SAM, 351, 358
Samuel, A. L., 21, 64, 203, 209, 885, 891, 1167
Samuelsson, C., 791, 1167
sándwich de jamón, 928
Sanna, R., 861, 1138
Sanskrit, 413, 937
Santorini, B., 973, 1154
SAPA, 519
Saraswat, V., 1173
Sastry, S., 63, 1143
3SAT, 157
satisfabilidad, **236**, 264, 458
satisfacción, **12**
  de restricciones, 23
    problema, 155, 155-159
Sato, T., 356, 605, 1167, 1171
SATPLAN, 459, **459**, 463-465 468, 469, 474, 521, 803
SAT-SOLVER, 459
saturación, **342**
Saul, L. K., 603, 605, 659, 1146 1167
Saund, E., 972, 1167
Savage, L. J., 540, 555, 691, 1167
Sayre, K., 1076, 1167
Scarcello, F., 356, 417,1140
Schabes, Y., 938, 974, 1165
Schaeffer, J., 148, 203, 211, 1131, 1154, 1162, 1167
Schank, R. C., 28, 939, 1167
Schapire, R. E., 768, 1138, 1167
Scharir, M., 1068, 1167
Schbning, T., 1167
Scheines, R., 859, 1170
Scherl, R., 415, 1060, 1152
Schickard, W., 7
Schmolze, J. G., 417, 1167
Schneeberger, J., 414, 1143
Schnitzius, D., 509, 1133
Schoenberg, L J., 861, 1158
Schofield, P. D. A., 99, 1167
Scholkopf, B., 861-863, 1131, 1133, 1167
Schöning, T., 265
Schoppers, M. J., 522, 1167
Schrag, R. C., 265, 1133
Schröder, E., 263, 1167
Schubert, L. K., 413, 468, 1139, 1145
Schultz, W., 892, 1167
Schulz, D., 1128
Schumann, J., 357, 1152
Schütze, H., 940, 973, 1154, 1167
Schwartz, J. T., 941, 1167
Schwartz, S. P., 413, 1167
Schwartz, W. B., 556, 1140
Scott, D., 604, 1168
Scriven, M., 1096, 1168
Searle, J. R., 14, 937, 1083, 1086-1090, 1097, 1168
secuencia de percepción, **38**, **40**
secuencial, 665, 681, 697

bajo incertidumbre, 530
circuito, 256
entorno, 47
problema de decisión 698-703, 733
segmentación, **990**
(de una imagen), **990**
(de una sentencia), 47
(del habla), 652
SEGMENTACIÓN-VITERBI, **948**
seguimiento de juguetes, **1031**
Segunda Guerra Mundial, 11, 602
seguro
de automóvil, 672
de vida, 672
Sejnowski, T., 862, 892, 1143, 1157, 1172
SELECCIÓN-ALEATORIA, **134**
SELECCIONAR-NO ASIGNADA, 163
Self, M., 860, 1129
Selfridge, M., 975, 1125
Selfridge, O. G., 20
Selman, B., 148, 264, 417, 468,1140, 1147, 1148, 1168
SEMÁNTICA, 904
semántica, 34, **225**
de composición, **919**
lógica, 261
semidecidible, 352
semiótica, **937**
Sen, S., 892, 1160
sensor, 38, 45, **979**
activo, **1025**, 501, 502, 979
de fuerza, **1027**
de imagen, **1026**
de sónar, **1025**
de toque, **1027**
fallo, 637
modelo, **614**, 624, 657, 713, 1030
pasivo, **1025**
preceptor, **1026**
táctil, 979
sensores táctiles, **1025**
sentencia
atómica, **229**, 281, 286, 302
como configuración física, 228
compleja, 229, 281, 302
compuesta, 281, 302
cuantificada, 302
en un lenguaje, **899**
en una BC, **219**, 261
sentencias
de observación, 8
derivadas, 227
sentido común, 595
separación (en red Bayesiana), 550
separación d, **569**
SEQUITUR, 934, 935, 940, 952
serendipia, 506
Sergot, M., 415, 1150
Serina, L, 468, 1139
sesgo, declarativo, **794**, **799**
Sestoft, P., 806, 1146
SETHEO, 358
Settle, L., 357, 1141
Seymour, P. D., 178, 1165
SGP, 468
Shachter, R. D., 569, 602, 604, 673, 691, 735, 1168, 112
Shafer, G., 604-606, 1168
Shahan, R. W., 1098, 1125
Shahookar, K., 99, 1168
Shakey, 64, 472, 521, 1066
Shalla, L., 357, 1176
Shanahan, M., 415, 1168
Shankar, N., 351, 1168
Shannon, C. E., 20, 182, 208, 750, 767, 972, 1155, 1168
Shapiro, E., 808, 1168
Shapiro, S. C., 35, 1168
Shapley, S., 736, 1168
Sharir, M., 1067, 1168
Sharp, D. H., 860, 1131
Shavlik, J., 769, 1168
Shaw, J. C., 99, 264, 1159
Shawe-Taylor, J., 861, 863, 1131
Shazeer, N. M., 33, 1153
Shelley, M., 1093, 1168
Shenoy, P. P., 606, 1168
Shewchuk, J., 1133 Shi, J., 990, 1168
Shieber, S. M., 941, 1162
Shimelevich, L. I., 657, 1176
Shin, M. C., 733, 1163
Shmoys, D. B., 99, 519, 1151, 1154
Shoham, Y., 64, 357, 415, 691, 1151, 1168, 1169
Shortliffe, E. H., 27, 605, 1128, 1169
SHRDLU, 24, 27, 355, 434
Shwe, M., 603, 1169
Sidner, C. L., 940, 1141
Siekmann, J., 358, 1169
Sietsma, J., 861, 1169
SIGART, 35
SIGIR, 975
signos, **897**
Siklossy, L., 1169
silogismo, 5, **353**
Silverstein, C., 960, 1169
Simard, P., 758, 897, 1152
símbolo, 229
constante, 278, 280, 302
de igualdad, **286**
de inicio, **1117**
de predicado, **278**, 302
no terminal, **900**, 901, 1117
puro, 248
terminal, **897**, 1117
Simmons, R., 100, 939, 1067, 1149, 1169
Simon, H. A., 4, 12, 20, 21, 34, 64, 99, 146, 202,264, 358, 466, 1104, 1151, 1159, 1169
Simon, J. C., 265, 1169
Simons, P., 417, 1159
simple-Bayes, **550**, 554, 556, 818, 829, 830, 955
SIMPLIFICAR, 349
simulación del mundo, 1085
sinapsis, 13
Sinclair, A., 1153, 1164
sincronización, **514**
Singh, M. P., 64, 1145
Singh, S. P., 733, 1123, 1145, 1147, 1171
singularidad
tecnológica, **1094**
sinónimos, 395, 957, **957**
sintaxis, 28, 225
edulcorada, **290**
lógica, 261
síntesis, **352**
de algoritmos, 351
deductiva, **352**
de programa, 521
SIPE, 518-522
Sirovitch, L., 1169
sistema
basado en reglas, 596, 1080
de lente, 983-985
de mantenimiento de verdad, 177, **409**, 49-411, 418, 1097
basado en justificaciones, **410**
basado en suposiciones, **411**, 418
de Posicionamiento Global (GPS), **1026**
diferencial, **1026**
de producción, 309, **325**, 352
Decimal Dewey, 367
esparcido, **566**
experto, 412, 686, 690, 1092
basado en Prolog, 329
comercial, 325

con incertidumbre, 30
decisión-teorético, 686-690
primero, 27
HPP (Programación Heurística, Proyecto), 27
lógica, 596
médica, 32, 604
primero comercial, 28
localmente estructurado, **566**
sistemas
dinámicos, 656
multiagente, 719
situación, **373**, 664
situado, 31
SKETCHPAD, 176
Skinner, B. F., 18, 63, 1169
Skolem, T., 303, 354, 1169
skolemización, 311, **337**
Slagle, J. R., 23, 209, 1169
Slate, D. J., 100, 1169
Slater, E., 208, 1169
Sleator, D., 938, 1169
Slocum, J., 939, 1169
Sloman, A., 1097, 1169
Slovic, P., 1147
Smallwood, R. D., 734, 1169
Smith, A. F. M., 657, 821, 860, 1124, 1140, 1172
Smith, A., 11
Smith, B., 63, 1146
Smith, D. E., 335, 358, 468, 520, 1139, 1162, 1169, 1170, 1174
Smith, D. R., 358, 1170
Smith, J. M., 149, 1170
Smith, J. Q., 691, 692, 1160, 1170
Smith, R. C., 1066, 1170
Smith, R. G., 1127
Smith, S. J. J., 1170
Smith, V., 736, 1164
Smith, W. D., 606, 1123, 1158
SMODELS, 417
Smola, A. J., 863, 1167
Smolensky, P., 29, 1170
Smyth, P., 657, 1170
SNARC, 19
SNLP, 467
SOAR, 31, 325, 355, 519, 806, 1102
sobreajuste, **754**, 753-755, 811, 851
sobregenera, **905**
sobregeneración, **905**
Sócrates, 4
Soderland, S., 467, 1170
softbot, **46**
SOLICITUD, 685
sólido triédrico, **1002**
Solomonoff, R. J., 20, 769, 1170
soltero, 368
solución, **69**, **71**, **98**, 444, 720
cíclica, 497
en planificación, **431**
óptima, **71**
SOLUTION, 81, 86, 93
soma, 13
sombreado, 992, 999-1000
Somers, H., 940, 974, 1131, 1145
Sompolinsky, H., 861, 1121
Sondik, E. J., 735, 11698, 1170
soneto, 1081
sonido (inferencia), 262
Sosic, R., 177, 1170
Soutchanski, M., 415, 1126
Sowa, J., 419, 1170
Sparck Jones, K., 556, 940, 1141, 1165
SPASS, 358
SPEECH-PART, 903
SPI (Inferencia Simbólica Probabilística), 602
Spiegelhalter, D. J., 602, 859, 1139, 1151, 1156, 1170
Spielberg, S., 1095, 1170
SPIKE, 519
SPIN, 352
Spirtes, P., 859, 1170
Springsteen, B., 961, 1170
Sproull, R. F., 692, 1136
Sputnik, 25
SQL, 403
SRGP, 4, 8, 21
SRI, 22, 303, 466, 691
Srinivas, B., 938, 1134
Srinivasan, A., 706, 709, 1160, 1170
Srivas, M., 1170
Srivastava, B., 1147
SSD, *véase* suma de diferencias al cuadrado, algoritmo SSS*, 211
Stader, J., 519, 1121
STAGE, 148
STAHL, 809
Stallman, R. M., 177, 1170
STAN, 468
Stanfill, C., 756, 1170
Stanford University, 20, 22, 26, 27, 303
Staniland, J. R., 556, 1132
States, D. J., 860, 1145
static ( ), 1118
STATLOG, 769
Steel, S., 519, 522, 1121
Stefik, M., 419, 605, 1170
Stein, J., 606, 1158
Stein, L. A., 1106, 1170
Steinbach, M., 959, 1170
Steiner, W., 1128
Stern, H. S., 863, 1139
Sternberg, M. J. E., 1148, 1170, 1172
Stevens, K. A., 1017, 1170
Stickel, M. E., 349, 358, 939, 975 1143, 1170
Stiller, L. B., 204, 1170
Stillings, N. A., 34, 1171
Stob, M., 769, 1160
Stockman, G., 211, 1171
Stokes, 1., 1121
Stolcke, A., 974, 1171
Stone, P. J., 768, 1127
Stone, P. T., 768, 1145
Stone, P., 522, 1171
Stork, D. G., 863, 1135
Story, W. E., 99, 1146
Strachey, C., 16, 211, 1171, 1172
Strat, T. M., 606, 1166
Striebel, C. T., 657, 1164
STRIPS, 429-432, 466,470-473, 475, 482, 514, 519-522, 734, 806
STRIPS assumption, 431
Strohm, G., 1137
Stuckey, P. J., 177, 278, 357, 1145, 1154
STUDENT, 23
Stutz, J., 860, 1129
suavizado, 619-622, 653, 656 **947**, 989
de adición unitaria, **946**
de interpolación borrado, 973
de intervalo fijo, **622**
Good-Turing, 973
online, 626
subasta
de Vickrey , 732
inglesa, **730**
SUBCADENA, 134
subcat, *véase* lista de subcategorización
subcategorización, **916**, 916-918
lista, 916
sub-evento, **381**
subjetivismo, **538**
subproblemas independientes, **171**
Subramanian, D., 419, 807, 1105, 1166, 1171
SUBST, 310, 327, 484

subsunción, 347
en lógica de enunciados, **404**
en resolución, **348**
retículo, 317
SUCCESSOR-FN, 70
SUCESOR, 186, 190, 496
Sueco, 36
Sugihara, K., 1171
Sugnet, C., 860, 1127
Sulawesi, 171
SUMA, 1109, **1109**, 1110
suma
de diferencias al cuadrado, **993**
de errores al cuadrado, **820**
Sun Microsystems, 1092
Superman, 273, 388
suposición, **411**
de mundo cerrado, 411, 430, 500
suposiciones de trasfondo, 992
Sussman, G. J., 177, 355, 467, 1170, 1171
sustancia, 371-373
espacial, **383**
temporal, **383**
sustancia temporal, *véase* ecuación de diferencias temporales
sustanciales, **874**
Sustantivo
contable, 372
no contable, **372**
sustitución, **287**, 310
Sutherland, I., 176, 1171
Sutherland, G. L., 1128
Sutton, R. S., 734, 892-894, 1123, 1171
Svartvik, J., 937, 1164
Svestka, P., 1068, 1147
Svetnik, V. B., 657, 1176
SVM, *véase* máquina de vector de apoyo
Swade, D. D., 16, 1171
Swartz, R., 1128
Swerling, P., 657, 1171
Swift, T., 356, 1171
Syrjánen, T., 417, 1159, 1170
SYSTRAN, 965, 975
Szathmáry, E., 149, 1170
Szeliski, R., 993

T(cat, i.) (evento líquido), 383
T(*s, a, s′*) (modelo de transiciones), 699, 870
T4, 519
tabla
de búsqueda, 851
de transposición, 191
hash, 317
Tadepalli, P., 709, 1171
TAG (Gramática de Colindancia de Árboles), 938
Tait, P. G., 99, 1171
taller de Dartmouth, 20
Talos, 1066
Tamaki, H., 356, 972, 1161, 1171
Tambe, M., 806, 1171
Tanca, L., 355, 1128
Tank, D. W., 1160
Tarjan, R. E., 1171
Tarski, A., 9, 303, 938, 1171
Tash, J. K., 734, 1172
Tasmania, 171
Tate, A., 467, 489, 518, 519, 1121, 1124, 1172
Tatman, J. A., 735, 1172
TAUM-METEO, 964, 975
taxi, 44, 45
automatizado, 61
en Atenas, 559
taxonomía, **367**, 413
Taylor, C., 769, 1017, 1133, 1156
Taylor, R., 1067, 1153
Taylor, W., 1129
TD, *véase* aprendizaje de diferencias temporales
TD-GAMMON, 204, 211
teléfono (habla), **645**
telescopio, 608
Espacial Hubble, 158, 170, 475, 519
televisión, 897
Teller, A., 1156
Teller, E., 1156
Temperley, D., 938, 1169
TEMPLE SIMULADO, **130**, 249
temple simulado, 139, 149, 153, 177, 590
tenis, 512, 516
de mesa, *véase* ping-pong
Tennenholtz, M., 892, 1126
teorema, **289**
de Herbrand, **344**, 354
de incompletitud, 9, 343, 1078
de representación, **677**
del límite central, **1115**
del perceptrón, 23
fundamental de la resolución, 244, 342
incompletitud, 9
teoría
de cadenas, 381
de control, 17, 63, 149, 466, 860, 886, 1053
adaptativo, 870, 891
óptima, 149
de decisión, 11, 31, 530, 690
de especificación funcional, 1096
de estructura retórica, 939
de identidad, 1096
de la confirmación, 8, 556
de la posibilidad, 605, **605**
de números, 809
de optimalidad (Lingüística), 939
de retículos, 351
de utilidad multiatributo, **674**, 691, 702
del aprendizaje computacional, **760**, **767**, **769**, **862**
Dempster-Shafer, 596, 598-599, 605
descriptiva, **672**
inversa de juegos, 729
normativa, **672**
sintáctica (del conocimiento), **388**
Teórico Lógico, 20, 264
TERMINAL?, 872, 874, 881
término
base, **282**, 310
complejo, 302
cuantificado, **923**, 923
(en lógica), 280, **280**
terremoto, 563
Tesauro, G., 204, 211, 883, 885, 893, 1172
TEST
CORTE, 194
-OBJETIVO, 81, 86, 93, 116, 141, 144, 496
-TERMINAL, 183, 190, 194
TETRAD, 859
Teukolsky, S. A., 1163
texel, **998**
texto ancla, **392**
textura, 992, **997**, 997
Thagard, P., 1172
Thaler, R., 608, 1172
THEO, 1102
Theseus, 767
Thielscher, M., 414, 1172
Thitimajshima, P., 1124
Thomas, A., 603, 1139
Thomason, R. H., 938, 1172
Thompson, D. W., 1008, 1172

Thompson, H., 1125
Thompson, K., 210, 1130
Thompson, S. A., 940, 1154
Throop, T. A., 1170
Thrun, S., 415, 1061, 1067 1126, 1128, 1137, 1157, 1172
ti y tú, 904
Tibshirani, R., 747, 1138, 1142
tiempo, 413
de mezcla, 619
de respuesta (en IR), **956**
Tinsley, M., 204
tipo de unión 1003
Tirole, J., 735, 1138
Titterington, D. M., 859, 1172
TL, 20
TMS, *véase* sistema de gestión de certezas
Toffler, A., 1092, 1172
Tomasi, C., 995, 1016, 1172
tomate, 649
Torrance, M. C., 1140
Torras, C., 520, 1146
Touretzky, D. S., 417, 1172
tracto vocal, 649
traducción, 36, 964-972
basada en memoria, 966
estadística, 967-972
TRADUCIR–A-SAT, 459
tragedia de los mancomunados, **730**
transductores de estado finito en cascada, **962**
transhumancia, **1094**
transición de fase, 265
transitividad (de preferencias), 667
transparencia referencial, 388
trasfondo, 1080, 1081
Traverso, P., 468, 1125, 1130
trazado de rayos (ray tracing), 868
TREC, 960
tres en raya (tic-tac-toe), 183, 208, 212
trigrama, **652**
TRIPLETA, 1012
troceado, 962
Troyanskii, P., 975
Trucco, E., 1018, 1172
Tsang, E., 178, 1172
T-SCHED, 519
Tsitsiklis, J. N., 734, 884, 892-894, 1116, 1125,1154, 1161, 1172
TSP, *véase* problema del viajante de comercio
Tubo de vacío, 20
Tung, F., 657, 1164
tupla, 278
turbo descodificación, **604**
Turco, 208
Turcotte, M., 706, 1172
Turing, A., 3, 10, 16, 19-21, 35, 59, 182, 209, 312, 354, 602, 860, 891, 967, 1076-1079, 1082, 1089, 1096, 1098, 1107, 1172
Turnen K., 736, 1172
Turtle, H. R., 974, 1172
Tversky, A., 568, 672, 1147, 1173
TWEAK, 467
Tyson, M., 974, 1143

U (utilidad), 665
ubicación de aeropuertos, 673, 678
UCPOP, 468
Ullman, J. D., 335, 938, 1115, 1121, 1123, 1164, 1173
Ullman, S., 1016, 1017, 1145, 1173
umbral, **840**, 843
UMG (unificador más general), 315, 317
unicornio, 267
unidad oculta, **841**
unificación, **314**, 312-318, 352
e igualdad, 345
unificador, más general (UMG). 315, **315**, 317, 344, 359
UNIFICAR, 315, 316, **316**, 320, 327, 330, 331, 345
-VAR, 316, **316**, 330
UNIR, 330 acción aplicable, **430**
Universidad Carnegie Mellon, 20
universo de Herbrand, **342**, 354
UNPOP, 468
UOSAT-II, 519
URP, 691
Urquhart, A., 414, 1164
Uskov, A. V., 209, 1121
Utgoff, P. E., 768, 1157
UTILIDAD, 185, 187, 190
utilidad, **58**, 182, 531
económica, 669-671
escala, 671-671
esperada, 64, 531, 664, 670
función, **59**, 183, 664, 668-674
independencia, **678**
multi-atributo, 674-678
multiatributo, 690
multiplicativa, **678**
nodo, **679**
normalizada, **673**
ordinal, **673**
principio, 668
teoría, **531**, 666, 690
utilitarismo, 8
utopía, 1107
UWL, 521
VACÍO, 80
Vaessens, R. J. M., 519, 1173
Valiant, L., 769, 1173
validación cruzada, 755, 851
Validez, **235**
VALOR, 126, 130
-MAYORÍA, 749
valor
de información perfecta, **683**
de la información, 682-686, 690, 695, 1081, 1104
esperado (en un árbol de juego) 199
material, **193**
monetario esperado, 669
por defecto, 400
van Beek, P., 176-178, 1122, 1149
van Benthem, J., 414, 1173
van Doorn, A. J., 1016, 1149
Van Emden, M. H., 356, 418, 1173
van Harmelen, F., 807, 1173
van Heijenoort, J., 358, 1173
Van Hentenryck, P., 177, 1173
van Nunen, J. A. E. E., 733, 1173
van Roy, B., 893, 1172-1173
Van Roy, P. L., 328, 332, 356, 1173
van Run, P., 161, 178, 1122
Vapnik, V. N., 769, 861-863, 1126, 1131, 1152, 1173
Varaiya, P., 63, 893, 1137, 1150
Vardi, M. Y., 356, 416, 1136, 1150
variabilización (en EBL), 787
eliminación, **577**, 577-579, 601, 603, 641
en lógica, 281
indicador, 828
ordenación, 162
¿VARIABLE?, 316
Varian, H. R., 736, 1173
vaso giratorio, 861
Vaucanson, J., 1066
Vazirani, U., 148, 769, 1121, 1148
Vecchi, M. P., 176, 1148
Vecino más cercano, **835**, 1008
VECINOS, 165
vector, **1112**
de alineamiento de palabras, **971**
de soporte, 852, 854

máquina, 769, **852**, 851-857, 863
virtual, 856
vehículo
aéreo no tripulado (UAV), **1024**
autónomo, 32
subacuático autónomo (AUV), 1024
terrestre no tripulado (ULV), **1024**
velocidad de muestreo, **647**
Veloso, M., 806, 1173
Vempala, S., 858, 1161
Ventaja material, 193
ventana de Parzen , 860
verdad, **225**, 281
preservar inferencia, **227**
tabla de, **231**, 263
¿VERDADERA-LP?, 232, 234, 266
Vere, S. A., 1173
verificación, **351**
de circuito, **301**
Verma, T., 602, 859, 1139, 1161
Verri, A., 1018, 1172
vértice
(de un gráfico), **908**
(poliedro), 1001
Vetterling, W. T., 1163
Vienna, 1083
Vigoda, E., 658, 1153
Vinge, V., 1094, 1173
Viola, P., 1018, 1173
visión, 3, 6, 14, 176, 981-1015
activa, 1081
a posteriori, **616**
por computador, *véase* visión
conclusión (de una implicación), **229**
Visser, W., 1142
Vitanyi, P. M. B., 769, 1152 algoritmo deViterbi, **624**
VLSI, 77, 131
Volk, K., 859, 1140
von Kempelen, W., 208
von Linne, C., 413
von Mises, R., 555, 1173
von Neumann, J., 11, 18, 19
von Neumann, J., 11, 208, 691, 735, 1173
von Stengel, B., 736, 1149
von Winterfeldt, D., 691, 1173
Voorhees, E. M., 975, 1173
Vossen, T., 468, 1173
VPI, *véase* valor de información perfecta
VUELTA-ATRÁS-RECURSIVA, 160
Wadsworth, C. P., 1140
Waibel, A., 659, 1173
Waldinger, R., 304, 357, 358, 467,1154, 1166, 1173
WALKSAT, 169, 251, **251**, 252-254, 265, 406, 412, 463, 465, 474
Walker, E., 33, 1131
Walker, H., 1140
Wall, R., 938, 1134
Wallace, D. L., 1158
Walras, L., 11
Walsh, M. J., 1137
Walter, G., 1066
Waltz, D., 23, 176, 860, 1004, 1170, 1173
WAM, *véase* Máquina abstracta de Warren, 356
Wang, E., 418, 1171
Wanner, E., 275, 1173
Warmuth, M., 99, 1125, 1164
WARPLAN, 467, 520
WARPLAN-C, 520
Warren, D. H. D., 328, 331, 356, 467, 520, 937, 1162, 1174
Warren, D. S., 356, 1171
Washington, G., 387
Wasow, T., 938, 1166
Watkins, C. J., 734, 1174
Watson, J., 15
Watson, J. D., 135, 1174
Watt, J., 17
Wattenberg, M., 149, 1146
Weaver, W., 750, 972, 975, 1168
Web Semántica, 413
Webber, B. L., 34, 940, 1141, 1146, 1174
Weber, J., 657, 993, 1144
Wefald, E. H., 147, 211, 1103, 1166
Wegbreit, B., 1066, 1157
Wegman, M. N., 357, 1161
Weidenbach, C., 358, 1174
Weiner, N., 975
Weinstein, S., 940, 1141, 1160
Weisler, S., 1170
Weiss, G., 64, 522, 1174
Weiss, S., 675, 975, 1174
Weiss, Y., 531, 1158, 1176
Weizenbaum, J., 1092, 1097, 1174
Weld, D. S., 64, 419, 467, 469, 519, 520, 1123, 1135, 1136, 1162, 1170, 1174
Wellman, M. R. 12, 604, 606, 691,734, 1068, 1133, 1174
Wells, P., 1134
Werbos, P., 734, 861, 892, 1174
Wermuth, N., 1151
Wertheimer, M., 1017
Wesley, L. P., 522, 1175
Wesley, M. A., 1068, 1174
Westinghouse, 518
Weymouth, T., 1067, 1130
Wheatstone, C., 1017, 1174
White, J. L., 1142
Whitehead, A. N., 19, 353, 1175
Whiter, A. M., 518, 1172
Whitney, R. A., 939, 1123
Whorf, B., 275, 1175
Widrow, B., 23, 861, 872, 1175
Widrow-regla de Hoff, **883**
Wiener, N., 18, 182, 208, 1165, 1175
Wilber, B. M., 357, 1166
Wilcox, B., 212, 1164
Wilczek, F., 1123
Wilensky, R., 28, 1175
Wilfong, G. T., 1066, 1131
Wilkins, D. E., 207, 518, 521, 1175
Wilks, Y., 939, 1175
Williams, B., 418, 519, 1158, 1159
Williams, R., 693
Williams, R. J., 733, 861, 889, 1162, 1165,1175
Williamson, M., 1136
Wilson, A., 973, 1152
Wilson, R. A., 4, 34, 1097, 1175
Wilson, R. J., 1125
Windows, 602
Winker, S., 351, 357, 1176
Winograd, S., 23, 1175
Winograd, T., 23, 27, 355, 1125, 1171, 1175
Winston, P. H., 2, 23, 1175
Wirth, R., 709, 1175
Witten, I. H., 934, 940, 973, 1159, 1175
Wittgenstein, L., 7, 228, 264, 370, 413, 937, 1175
Wóhler, F., 1083
Wojciechowski, W. S., 352, 1175
Wojcik, A. S., 352, 1175
Wolf, A., 938, 1141
Wolpert, D., 736, 1172
Wong, A., 973, 1167
Wood, D. E., 147, 1151
Wood, M. K., 149, 1175
Woods, W. A., 416, 938, 1175, 1176
Wooldridge, M., 64, 1176
Woolsey, K., 886
Wordnet, 975

World Wide Web, 31, 391, 945, 952, 958, 976
Wos, L., 351, 357, 1176
Wright, O. y W., 3
Wright, S., 149, 1176
Wrightson, G., 358, 1169
Wu, D., 1176
Wundt, W., 14, 1016
Wygant, R. M., 355, 1176

XCON, 325
Xerox, 964
XML, 413, 963
xor, *véase* OR exclusivo
XTAG, 938

Yakimovsky, Y., 692, 1136
Yale, 28
Yamada, K., 1139, 1176
Yan, D., 1138
Yang, C. S., 974, 1167
Yang, Q., 469, 519, 1176
Yannakakis, M., 150, 1124, 1161
Yap, R. H. C., 356, 1145
Yedidia, J., 604, 1176
Yip, K. M.-K., 418, 1176
Yngve, V., 938, 1176
Yob, G., 266, 1176
Yoshikawa, T., 1068, 1176
Young, R. M., 519, 1176
Young, S. J., 973, 1151
Young, T., 868
Younger, D. H., 1176
Yung, M., 99, 1121, 1142
Yvanovich, M., 519, 1133

Z-3, 16
Zadeh, L. A., 605, 1176
Zaidel, M., 938, 1134
Zapp, A., 1018, 1134
Zaritskii, V. S., 658, 1176
Zelle, J., 940, 1176
Zermelo, E., 735, 1176
Zhai, C., 974, 1151
Zhang, L., 1158
Zhang, N. L., 602, 1177
Zhang, W., 148, 1150
Zhao, Y., 1158
Zhivotovsky, A. A., 1121
Zhixing, C., 205
Zhou, R., 148, 1177
Zhu, D. J., 1067, 1177
Zilberstein, S., 520, 1103, 1141, 1177
Zimmermann, H.-J., 605, 1177
Zisserman, A., 1017, 1137, 1142, 1158
Zlotkin, G., 1165
Zobrist, A. L., 212, 1167
Zog, 784
Zuckerman, D., 1153
zumbido, **1063**
Zuse, K., 16, 1177
Zweben, M., 519, 1133
Zweig, G., 1177
Zytkow, J. M., 1151

2ª Edición

Inteligencia Artificial
Un Enfoque Moderno

Russell • Norvig

La primera edición de ***Inteligencia Artificial: Un Enfoque Moderno*** se ha convertido en un clásico de la literatura sobre la IA. Ha sido adoptado por 600 universidades de 60 países y ha recibido elogios por ser una síntesis definitiva de este campo. Se ha dicho por ejemplo que:

"La publicación de este libro de texto fue un avance importante, no sólo para la enseñanza en el campo de la IA, sino también por el punto de vista unificado que introduce. Incluso, los expertos encuentran que todos los capítulos realizan importantes incursiones en la investigación de este campo." _Prof. Thomas Dietterich (Estado de Oregón)
"Es simplemente magnífico. El libro que siempre he estado esperando… es como la Biblia de la IA para la próxima década." _Prof. Gerd Brewka (Viena)
"Un logro maravilloso, un libro verdaderamente precioso." _Prof. Selmer Bringsjord (RPI)
"Es un gran libro, con una profundidad y una amplitud increíbles, y muy bien escrito. Sé que todas las personas que lo han utilizado en sus clases han disfrutado con él." _Prof. Haym Hirsh (Rutgers)
"Me ha impresionado profundamente esa calidad sin precedentes al presentar el campo de la IA con una imagen coherente, equilibrada, extensa y profunda. Se va a convertir en el libro de texto estándar durante los próximos años." _Prof. Wolfgang Bibel (Darmstadt)
"¡Es magnífico! Está bien escrito, tiene una estructura muy buena, y disfruta de toda la cobertura informativa que debería conocer todo estudiante de IA." _Prof. Martha Pollack (Michigan)
"Excepcional… Las descripciones son verdaderamente claras y legibles, y su estructura es excelente; los ejemplos son motivadores y su cobertura informativa es académica y profunda. Va a dominar su campo durante un tiempo por méritos propios." _Prof. Nils Nilsson (Stanford)
"El mejor libro de que disponemos por ahora… Es casi tan bueno como el que escribimos Charniak y yo, pero más actualizado. (De acuerdo, lo admitiré, puede que incluso sea mejor que nuestro libro.)" _Prof. Drew McDermott (Yale)
"Un informe magistral de gran alcance en el campo de la Inteligencia Artificial que iluminará a profesores, así como a alumnos." _Dr. Alan Kay

"Este es el libro que me hizo amar la IA." _Alumno (Indonesia)

En la segunda edición, todos los capítulos se han reescrito de forma extensa. Se ha introducido material nuevo y elocuente para abarcar áreas tales como la satisfacción de las limitaciones, gráficos de planificaciones, agentes de Internet, inferencia de probabilística exacta, técnicas Monte Carlo de la cadena Marlon, filtros Kalman, métodos de aprendizaje de conjuntos, aprendizaje estadístico, modelos de lenguajes naturales probabilísticos, robótica probabilística y aspectos éticos de la IA.

El libro tiene un soporte variado de recursos en línea (*online*) incluyendo códigos fuente, figuras, diapositivas para clase, un directorio de 800 enlaces con la IA en la Web y un grupo de debate en línea (*online*). Toda esta información se encuentra disponible en:

aima.cs.berkeley.edu

www.librosite.net/russell
LibroSite

LibroSite es una página web asociada al libro, con una gran variedad de recursos y material adicional tanto para los profesores como para estudiantes. Apoyos a la docencia, ejercicios de autocontrol, enlaces relacionados, material de investigación, etc., hacen de LibroSite el complemento académico perfecto para este libro.

www.librosite.net/russell

PEARSON
Educación

www.pearsoneducacion.com

ISBN 978-84-205-4003-0
9 788420 540030